VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza, návrh a implementace globálního provizního systému

Analysis, design and implementation of a global affiliate systém

2017 Vojtěch Oczka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Vojtěch Oczka**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Analýza, návrh a implementace globálního provizního systému
Analysis, Design, and Implementation of Global Affiliate System

Jazyk vypracování: čeština

Zásady pro vypracování:

Provizní systém spočívá ve vytvoření a správě sítě partnerů, kteří budou propagovat a prodávat produkty třetích stran (e-shopů) za přesně dané procentuální odměny. Cílem práce je analyzovat, navrhnout a implementovat webový provizní informační systém pro provozovatele eshopu, poskytovatele obsahu i běžné uživatele eshopu. Systém bude nabízen provozovatelům eshopů formou předplacené služby.

Úkoly:
1. Proveďte analýzu, návrh a implementaci globálního provizního systému.
2. Zpracujte rešerši cloudových databázových systémů a vybraný databázový systém použijte v implementovaném informačním systému.
3. Proveďte optimalizaci systému pro statisíce paralelních přístupů.
4. Navrhněte intuitivní rozhraní pro správce obsahu, provozovatele e-shopů a administrátory optimalizované pro mobilní zařízení.
5. Zpracujte administrátorskou i uživatelskou dokumentaci systému.
6. Zhodnoťte výsledný systém a porovnejte s podobnými aplikacemi.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jiří Kubica**

Konzultant diplomové práce: doc. Ing. Michal Krátký, Ph.D.

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení Studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. 6. 2017

.....

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava.

V Ostravě dne 22. března 2017

Railsformers s.r.o.
IČ: 24704440 DIČ: CZ24704440
www.railsformers.com
info@railsformers.com

.....
Podpis zástupce

Poděkování

Rád bych poděkoval panu doc. Ing. Michalovi Krátkému, Ph.D. za cenné rady, konzultace a odborný dohled při vedení mé diplomové práce. Dále bych rád poděkoval Ing. Jiřímu Kubicovi za možnost mít tuto práci ve firmě Railsformers s. r. o. a také Richardu Lapišovi za jeho rady během vývoje aplikace. Také bych rád poděkoval své přítelkyni Lence Pogatsové, mé milované dceři Valerii Oczkové a celé mé rodině za podporu během této práce.

Abstrakt

V této diplomové práci jsem vytvořil webový provizní informační systém dle zadání, které bylo vytvořeno firmou Railsformers s. r. o. Provedl jsem pro něj analýzu, návrh a následnou implementaci v jazyce Ruby on Rails. Tato aplikace využívá MySQL DBS a pro zjednodušení nasazení jsem ji propojil s Dockerem. Také jsem provedl rešerši databázových systémů v cloudu pro použití v produkčním nasazení. Poté jsem se zaměřil na optimalizaci aplikace, kde jsem nejdříve popsal a vybral vhodné nástroje, vytvořil generátor zatížení a pokusil se o samostatnou optimalizaci. Následně jsem zatížil aplikaci a porovnal jsem optimalizované výsledky s neoptimalizovanými, kde šlo vidět určité zlepšení. Nakonec jsem porovnal tento provizní systém s ostatními systémy na trhu a také vytvořil pro tento systém dokumentaci.

Klíčová slova

webový provizní informační systém, MySQL, Ruby on Rails, cloud, cloudový DBS, Gatling, Docker, reverzní proxy

Abstract

In this thesis I created a web affiliate information system by assignment which was created by company Railsformers s. r. o. I made conducted analysis, design and subsequent implementation in the Ruby on Rails programing language. This application uses MySQL DBS. I connected this aplication with Docker to easier deployment. I also compared MySQL DBS in the cloud to use in production. Then I focused on optimizing the application where in the first I described and selected the appropriate tools, created a performance generator and I tried to optimize this application. Subsequently, I loaded the application and compared the optimized results with the non-optimized where there was some improvement. Finnally I compared this affiliate system with others systems on the czech market and I made documentation for this system.

Key Words

web affiliate information system, MySQL, Ruby on Rails, cloud, cloud DBS, Gatling, Docker, reverse proxy

Seznam použitých zkratek a symbolů

- SQL: Structured Query Language
- UTM: Urchin Tracking Module
- URL: Uniform Resource Locator
- PNG: Portable Network Graphics
- GIF: Graphics Interchange Format
- XML: Extensible Markup Language
- RAM: Random Access Memory
- VPN: Virtual private network
- CSV: Comma-separated values
- IOPS: Input/output operations per second
- SSD: Solid-state drive
- CPU: Central processing unit
- HTML: HyperText Markup Language
- GC: Garbage collection
- SOAP: Simple Object Access Protocol
- HTTP: Hypertext Transfer Protocol
- HTTPS: Hypertext Transfer Protocol Secure
- REST: REpresentational State Transfer
- FTP: File Transfer Protocol
- JDBC: Java Database Connectivity
- LDAP: Lightweight Directory Access Protocol
- SMTP: Simple Mail Transfer Protocol
- IMAP: Internet Message Access Protocol
- TCP: Transmission Control Protocol
- DBS: databázový systém

Seznam ilustrací a seznam tabulek

Obsah

1 Úvod

V době, během které jsem si měl vybrat téma diplomové práce, jsem přemýšlel nad tím, co bych v této práci rád dělal a jaké téma by pro mě mohlo být zajímavé. Na začátku mě žádné téma nenapadlo, a tak jsem se podíval na témata, která byla vypsaná naší univerzitou. Z těchto témat mne žádné nezaujalo na tolik, abych si nějaké zvolil. Bylo zde sice několik zajímavých témat, ale žádné mě nepřesvědčilo o tom, že je pro mě tím pravým. Proto jsem se rozhodl, že tato témata si nechám jako poslední možnost v případě, že bych si žádné vhodné téma nevymyslel, nebo nenarazil na něco co by mne hned oslovilo. Vymyslet vlastní téma jsem se snažil dlouho dobu. Stále se mi však nedařilo vymyslet žádné vhodné téma, a tak jsem se začal pomalu smiřovat s tím, že si budu muset vybrat jedno z témat, které vypsala univerzita a mě alespoň trošku zaujalo. Na poslední chvíli mi však bylo nabídnuto téma pro mě dost zajímavé, a měl jsem ihned jasno v tom co si vyberu. Na začátku toho všeho jsem nevěděl úplně všechny důležité informace. To mi ale nevadilo. Tématem, které jsem si zvolil, byl provizní systém určený na zvýšení počtu objednávek jednotlivých e-shopů. Toto téma mi bylo nabídnuto Ing. Jiřím Kubicou, který vlastní firmu Railsformers s. r. o., která se zabývá tvorbou webových aplikací v jazyce Ruby on Rails [1]. Díky tomu, že jsem měl se zmíněnou firmou již kladné zkušenosti, vzniklé na základě stáží a pak následnou bakalářkou praxí, tak jsem se pro toto téma rozhodl prakticky ihned. Toto rozhodnutí výběru tématu bylo proto, že mi přišlo výhodné propojení školní části tvořené diplomovou prací s reálným životem tvořeným projektem ve firmě. Díky zmíněnému tématu jsem si mohl také ověřit získané dovednosti během studia i ve firemním prostředí a také získat nějakou další praxi, která je důležitá hlavně po škole, při hledání nové práce, velmi ceněna.

Jak jsem se už zmínil, tak tématem této diplomové práce je webový provizní informační systém, kde hlavním cílem tohoto systému je se pokusit nabídnout uživatelům jednoduchou možnost si přivydělat propagací produktů a obchodníkům s e-shopy dát jednoduchou a levnou možnost propagace svých produktů, a v neposlední ne méně důležité řadě, vytvořit firmě nový druh příjmu tvořeného pomocí této webové aplikace. V této práci jsem měl navrhnout a implementovat tento systém, následně se ho pokusit optimalizovat pro vysokou zátěž. Dále jsem měl provést rešerši cloudových databázových systémů a následně jeden z nich vybrat pro tento provizní informační systém. Kromě toho tento provizní informační systém má mít intuitivní rozhraní pro všechny typy uživatelů, jenž ho budou využívat. Těmito uživateli jsou správci obsahu, provozovatele e-shopů a administrátoři, kteří budou mít plnou kontrolu nad tímto provizním informačním systémem. Pro všechny tyto uživatele by orientace a následné ovládání mělo být snadné a přehledné. Kvůli aktuální době, ve které se začíná rychlým tempem přecházet z klasických stolních počítačů nebo notebooků na mobilní přenosná zařízení tvořená hlavně mobilními telefony a tablety, tak je potřeba, aby tento systém byl pro tyto zařízení plně připraven a mohl uživatelům nabídnout snadnou správu z těchto zařízení. V neposlední řadě jsem měl pro tento systém vytvořit uživatelskou a administrátorskou dokumentaci, která usnadní práci v začátcích během seznamování se s touto webovou aplikací. V závěru práce jsem pak měl ještě zhodnotit výsledek této práce a porovnat tuto webovou aplikaci s ostatními podobnými aplikacemi na současném webovém trhu.

V kapitole 2 je popsána specifikace a požadavky na informační systém. V následující kapitole 3 je provedena analýza aplikace. Následuje kapitola 4, která popisuje funkční analýzu, kde jsou popsány jednotlivé funkce pro tento webový provizní informační systém. Poté je kapitola 5 obsahující návrh systému. Další kapitola 6 pak vysvětluje, co to je to DBS v cloudu a následně se provádí srovnání největších poskytovatelů této služby, kde jsem se zaměřil pouze na typ DBS MySQL. Poté následuje

kapitola 7, která popisuje samotnou implementaci a strukturu testovacího nasazení aplikace. Následně je kapitola 8, která je zaměřena na testovací nástroje určené pro optimalizaci a výkonové zatížení aplikace. Je zde také projednáváno o vytvoření generátoru zatížení. V další kapitole 9 je pak popsána samotná optimalizace, která je rozdělena na optimalizaci pro vytváření testovacích dat, samostatné aplikace a aplikačního serveru. Další kapitola 10 pak projednává o samotném zatížení aplikace a porovnání výsledků optimalizované aplikace s neoptimalizovanou. Poté je kapitola 11, která porovnává výslednou webovou provizní aplikaci s ostatními systémy na trhu. Následně je kapitola 12, jenž zhodnocuje aplikaci a výslednou práci. Poslední kapitola 13 pak obsahuje použitou literaturu.

2 Specifikace a požadavky na informační systém

Tento systém je určen pro běžné uživatele a majitele e-shopů. Pro tyto uživatele by měla být práce v tomto systému co nejjednodušší. Dále by měl systém umožnovat registraci uživatelů s možností si vybrat typ využití provizního systému. Prvním typem využití je možnost přidání vlastních e-shopů nabízející provizi za uskutečněné nákupy. Dále musí být umožněno k těmto e-shopům jednoduše přidávat bannery s danou URL adresou pro následnou propagaci pomocí ostatních uživatelů. Druhým typem je pak výběr preferovaných e-shopů ze systému nabízející provizi za uskutečněný nákup.

Systém musí jednoznačně zpracovat příchozí požadavky vytvořené zákazníky, a to při kliknutí na daný banner nebo odkaz. U těchto požadavků musí systém z URL adresy zjistit uživatele, který doporučil zákazníka a URL adresu e-shopu na kterou se má daný zákazník přesměrovat. Tyto požadavky je nutné zaznamenávat kvůli možnosti identifikování uživatele a pro možné přiznání provize, v případě, že v daném e-shopu vznikla jakákoli objednávka. Při zpracovávaní požadavků můžou vzniknout dva typy chyb. Prvním typem je, že daný uživatel, který zákazníka doporučil, neexistuje nebo je zablokovaný. V tomto případě je zákazník přesměrován na URL adresu daného e-shopu, kde provize z případného nákupu zákazníka připadá rovnou proviznímu systému. Jestliže, nastane druhý typ chyby, který je způsoben neexistujícím e-shopem, je uživatel přesměrován na chybovou stránku provizního systému s oznámením, že daný e-shop v databázi provizního sytému neexistuje. Uživateli, který bude propagovat e-shopy, bude zobrazována přehledná statistika s přesným počtem přesměrování na e-shopy a také přehled objednávek. Na stránce s přehledem objednávek budou zobrazeny jednotlivé objednávky u daných e-shopů ve stavu zpracování. Pro administrátora bude systém umožnovat celkovou správu systému.

2.1 Rekapitulace požadavků systému

Systém umožní:

- Umožnění registrace s možností výběru typu využití systému.
- Možnost přidání nového e-shopu.
- Přidání bannerů k e-shopu s URL adresou.
- Výběr uživatelem preferovaných e-shopů k doporučování nákupu.
- Správné zpracování příchozích požadavků s následným přesměrováním na e-shop.
- Zaznamenávání příchozích požadavků.
- Reagování na možné chyby při zpracování požadavků.
- Zobrazení přehledné statistiky přesměrování na e-shopy a objednávek vztahující se pro daného uživatele.
- Zobrazení objednávek se stavem zpracování.
- Administrátorům umožnění celkové správy a přehledu provizního systému.
- Uživateli zobrazení získané provize.
- Jeden identifikující kód pro uživatele.
- Platnost cookies 30 dnů.
- Bannery musí mít omezenou velikost, která je definována, doporučenou velikostí od Googlu a Spir.cz.
- Systém musí mít podporu více jazyků.

- Jednotlivá doporučení budou vždy na celý nákup a z něho bude získaná následná provize pro uživatele, který přivedl daného zákazníka, který uskutečnil nákup.
- Optimalizace pro vysoký počet požadavků.
- Využití cloudové verze SQL od Googlu.
- Responzivní vzhled a optimalizace administrace pro mobilní telefony.

3 Analýza systému

3.1 Vize

3.1.1 Účel

Účelem systému je zajistit nové zákazníky pro registrované e-shopy, a to propagací jejich produktů pomocí registrovaných uživatelů nabízejících jejich produkty, ať už jakoukoliv cestou, za účelem získáni provize od e-shopu za uskutečněnou objednávku. Propagujícím uživatelům, se na úvodní stránce bude zobrazovat přehledný graf, který zobrazuje úspěšnost jejich propagace.

3.1.2 Obsluha

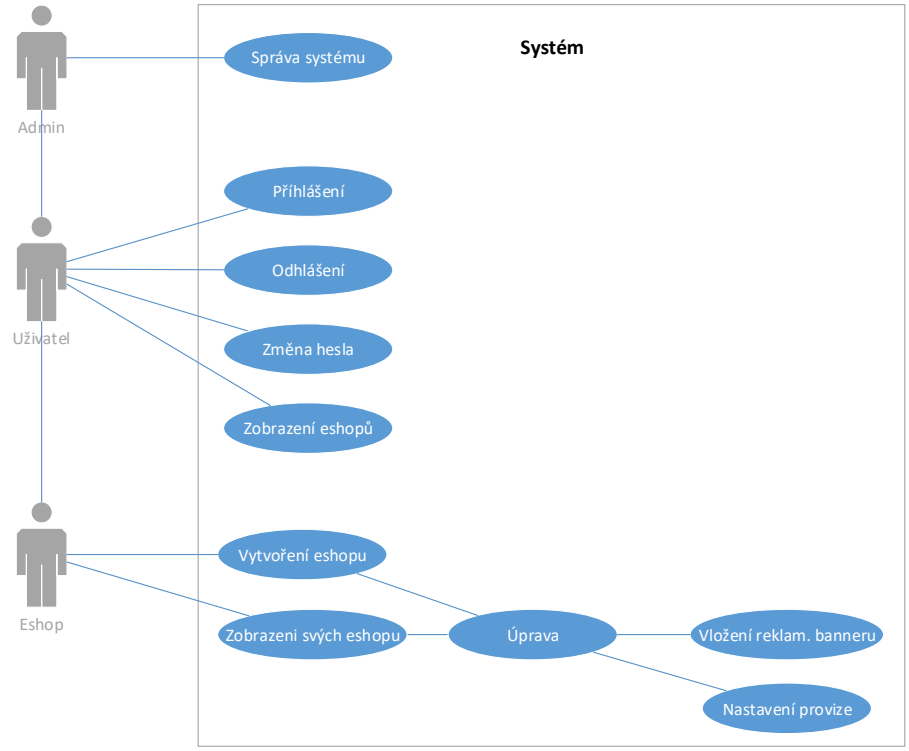
Systém bude obsluhovat více druhů uživatelů. Prvním je administrátor, který se stará o chod celého systému a má nad systémem plnou kontrolu. Dalším typem je e-shop, který do systému přidává své bannery a nastavuje si nabízenou provizi za uskutečněnou objednávku. Posledním typem je pak uživatel, který vidí jednotlivé e-shopy a také jejich nabízený reklamní materiál. Kromě toho vidí nabízenou provizi za uskutečněnou objednávku od e-shopu, podle které si může vybírat preferované e-shopy.

3.1.3 Komunikace se systémem

Systém bude obsluhován pouze přes webové rozhraní pro všechny typy uživatelů. Toto rozhraní by mělo být co nejvíce jednoduché a přehledné pro snadnou obsluhu jednotlivými typy uživatelů.

3.2 Diagram případů užití (Use Case diagram)

Obrázek 1 představuje diagram případů použití.



Obrázek 1: Diagram případů užití (Use Case diagram)

3.3 Výběr velikosti bannerů

Při analýze výběru velikosti reklamních prvků jsem vybíral doporučené velikosti od dvou významných služeb, kterými jsou Google Adsense [2] a Spir.cz [3]. Google, jak všichni víme, je celosvětová firma, která začala vytvořením internetového vyhledávače, který je nyní téměř nepostradatelný při vyhledávaní. Nyní se tato firma angažuje v rozmanitém segmentu profesí a služeb. Například v internetové reklamě, kde nabízí doporučené reklamní formáty, které jsou nejúspěšnější.

Spir.cz je české sdružení pro internetový rozvoj, které nabízí doporučení rozměrů reklamních ploch a jejich velikost. Jednotlivé rozměry lze vidět níže v protipříkladu s ostatními rozměry. Typy obrázků banneru byly zvoleny JPEG, PNG, GIF.

Doporučené rozměry		**Ostatní rozměry**	
Google Adsence	**Spir.cz**	**Google Adsence**	**Spir.cz**
• 300×250 • 336×280 • 728×90 • 300×600 • 320×100	• 728x90 Leaderboard 40 KB • 250x250 Square 30 KB • 120x600 Skyscraper 40 KB • 300x250 Medium Rectangle 40 KB • 468x60 Full Banner 20 KB	• 320×50 • 468×60 • 234×60 • 120×600 • 120×240 • 160×600 • 300×1050 • 970×90 • 970×250 • 250×250 • 200×200 • 180×150 • 125×125	• 88x31 • 120x60 • 120x90 • 125x125 • 468x60 • 234x60 • 700x100, 728x90, 745x100, 750x100 • 970x100, 998x100 • 728x120, 728x180, 728x200, 745x200 • 750x200 • 250x250, 300x300 • 640x480, 800x600 • 120x150, 120x300, 180x150 • 300x250, 480x300, 500x300 • 120x600 • 160x600 • 300x500, 300x600 • 300x250 • 240x400 • 120x240

3.4 UTM parametry

UTM parametry (Urchin Tracking Module) jsou označovány textové řetězce, které jsou připojovány k URL adrese. Tyto parametry nesou informace, které pak využívají analytické nástroje (např. Google Analytics [4]. Díky těmto parametrům můžeme od sebe odlišit jednotlivé zdroje návštěvnosti (odkud na web chodí lidé) a vyhodnocovat jejich účinnost. Detailní popis UTM parametrů je uveden zde. [5; 6] Jednotlivé UTM parametry a jejich význam je vysvětlen v tabulce 1.

UTM parametr	Význam UTM parametru
utm_source	Slouží k identifikaci zdroje. Například pro určení vyhledavače, názvu zpravodaje nebo jiného zdroje. Jednoduše řečeno název webu, ze kterého odkaz vede.
utm_medium	Identifikace použitého media (druh marketingového kanálu), například: • cpc = placené vyhledávání • organic = neplacené vyhledávání • referral = odkaz atp.
utm_term	Klíčové slovo, používající pro placené vyhledávaní. Slouží pro záznam klíčových slov dané reklamy.
utm_content	Slouží pro porovnání dvou reklam cílených na obsahovou síť. Slouží pro odlišení reklam nebo odkazů ukazující na stejnou adresu.
utm_campaign	Slouží k identifikaci určitého produktu nebo kampaně. Například se používá datum.

Tabulka 1: Popis jednotlivých UTM parametrů

Příklad adresy s použitím UTM parametrů:

www.adresa.cz/?utm_source=facebook&utm_medium=ads&utm_content=reklama-1&utm_campaign=kampan-1

3.5 Základní princip provizního systému

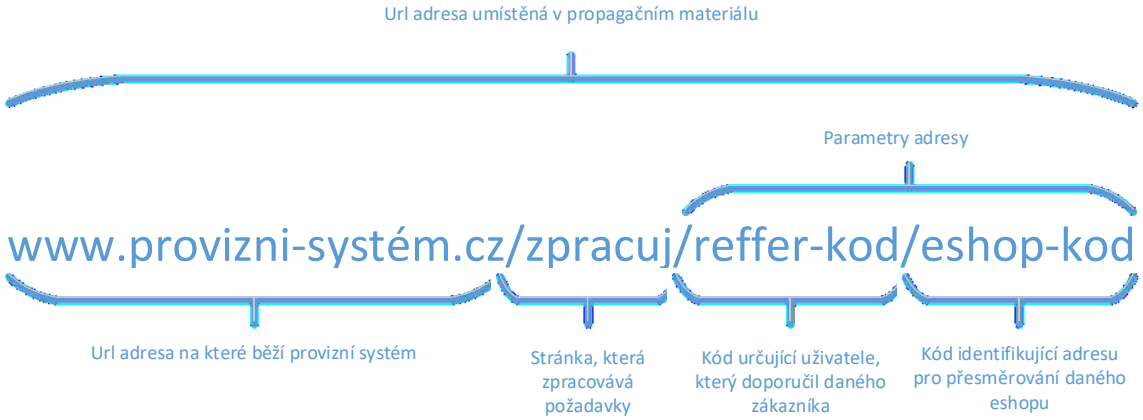
Proto, aby systém mohl správně fungovat, je zapotřebí, aby v systému byly zaregistrovaní dva typy uživatelů. Prvním typem jsou uživatelé, kteří budou mít na svém účtu přidané své e-shopy s nabízenými provizemi za uskutečněné nákupy, a také nejlépe nějaké reklamní bannery. Druhým typem uživatelů jsou ti, kteří následně budou tyto bannery nebo odkazy nějakým způsobem propagovat, aby přivedli potenciálního zákazníka. Následným kliknutím zákazníka na propagační materiál, je přiveden na náš provizní systém, který ho zpracuje a následně přesměruje na daný e-shop. Zpracování požadavku probíhá takovým způsobem, že systém díky URL adrese, která je uvedena na obrázku 2, zpracuje adresu, ze které získá jednotlivé parametry, podle kterých je zjištěn uživatel, který zákazníka doporučil, a URL adresa e-shopu, na kterou je zákazník přesměrován s přidanými UTM parametry, které jsou popsány v tabulce 2. Tyto přidané UTM parametry slouží pro identifikaci našeho systému na straně e-shopu a identifikaci uživatele, který zákazníka doporučil. Jakmile se zákazník dostane na stránku s e-shopem, tak je mu vytvořena cookies v počítači na určitou dobu, například 30 dnů. Cookies se vytváří na straně e-shopu proto, že z důvodu bezpečnosti, není možné vytvořit cookies u nás pro libovolně jinou stránku. Jestliže zákazník během této doby něco na e-shopu zakoupí, tak je nám toto sděleno pomocí XML souboru s objednávkami na straně e-shopu, který je generován a následně provizním systémem

načten jednou denně. Uživateli, který daného zákazníka doporučil, je následně připsaná jeho provize, kterou zaplatil daný e-shop za provedený nákup.

source	provizne.cz
medium	referral
campaign	xxxxxxx
content	zatím volné

Tabulka 2: Přidávané UTM parametry k adrese

Kde xxxxxxx je vygenerovaný 256bitová hashovaná hodnota. Pomocí této hodnoty se určuje daný záznam z přeměřování. Z tohoto záznamu se pak zjišťuje uživatel, který daného zákazníka doporučil.



Obrázek 2: Ukázka provizní adresy

4 Funkční analýza

4.1 Seznam funkcí

4.1.1 Evidence e-shopů

a. Vytvořit e-shop
b. Úprava vlastního e-shopu
c. Odstranění vlastního e-shopu
d. Zobrazení svých e-shopů
e. Zobrazení všech ověřených e-shopů
f. Přidáni do oblíbených
g. Odebrání z oblíbených
h. Zobrazení oblíbených e-shopů
Zodpovědnost: Uživatel
i. Zobrazení smazaných e-shopů
j. Obnovení smazaného e-shopu
k. Trvalé smazání e-shopu
l. Ověření e-shopu
Zodpovědnost: Správce
m. Import objednávek

4.1.2 Evidence bannerů

a. Přidaní banneru
b. Odstranění vlastního banneru
c. Zobrazení všech bannerů
d. Úprava vlastního banneru
Zodpovědnost: Uživatel

4.1.3 Evidence uživatelů

a. Přidat uživatele
b. Zablokovat uživatel
c. Přiřadit roli
Zodpovědnost: Správce
d. Upravit uživatele
Zodpovědnost: Správce, Uživatel
e. Registrace uživatele

4.1.4 Správa objednávek

a. Zobrazení doporučených objednávek
Zodpovědnost: Uživatel
b. Zobrazení všech objednávek
c. Editace stavů objednávky
Zodpovědnost: Správce
d. Přidat objednávku

4.1.5 Správa článků

a. Přidat článek

b. Odstranění článku
c. Upravit článek
Zodpovědnost: Správce
d. Zobrazení článku

4.1.6 Správa stránek

a. Přidat stránku
b. Odstranění stránky
c. Upravit stránku
Zodpovědnost: Správce
d. Zobrazení stránky
e. Kontaktní formulář

4.1.7 Přesměrování

a. Zobrazení přesměrování pro vlastní e-shopy
Zodpovědnost: Uživatel
b. Zobrazení všech přesměrování
Zodpovědnost: Správce

4.1.8 Přehled

a. Zobrazení částky k proplacení
b. Zobrazení uskutečněných objednávek
c. Zobrazení e-shopů s přehledem, jak si jednotliví uživatelé vedou
Zodpovědnost: Uživatel

4.1.9 Další funkce

a. Přidej hash kód
b. Zpracuj zákazníka (Get_and_redirect)
c. Import objednávek
d. Měsíční přehled

4.2 Detailní popis netriviálních funkcí

4.2.1 Přidej hash kód

Tato funkce je detailní popis funkce zmíněné v kapitole 4.1.9 v seznamu jako položka a. Tato funkce generuje hash hodnotu o délce 256 znaků z aktuálního času v milisekundách. Tento kód je ořezán na 16 znaků a uložen následně do databáze. Tato funkce je určena pro vytvoření hash kódu pro e-shop, uživatele a banner a je volaná v tom případě, pokud daný objekt ještě hash neobsahuje.

4.2.2 Zpracuj zákazníka

Tato funkce je detailní popis funkce zmíněné v kapitole 4.1.9 v seznamu jako položka b. Pomocí této funkce je zajištěno zpracování příchozích URL požadavků od zákazníků a následné přesměrování na konkrétní URL adresu e-shopu. Tato funkce pracuje takovým způsobem, že nejdříve získá příchozí parametry (hash uživatele a banneru) z URL adresy a následně pomocí těchto hashů najde v databázi odpovídajícího uživatele a banner. Dále vytvoří archivní záznam tohoto požadavku a to hlavně s aktuálním datem, uživatelem, bannerem a hashem identifikující tento záznam. Posledním krokem je

přesměrování uživatele na konkrétní URL adresu e-shopu získanou z banneru s danými UTM parametry popisovanými výše.

4.2.3 Import objednávek

Tato funkce je detailní popis funkce zmíněné v kapitole 4.1.9 v seznamu jako položka c. Tato funkce má na starosti importování uskutečněných objednávek z e-shopu v XML tvaru. Při importu se připojuje na adresu uvedenou v položce xml_import e-shopu. Následně stáhne aktuální XML soubor na server, kde ho uloží ve tvaru *„eshop_´id eshopu´_day_´aktualní den´.xml“* do složky *„private/xml_import/eshop_´id eshopu´/rok/mesic“*. Poté se pokusí naimportovat a uložit jednotlivé objednávky, které obsahují nějaké cookie shodující se s cookie ze zaznamenaných návštěv pro daný e-shop. Objednávky mají defaultní stav *„new“*. Jestliže se import nepodaří, navrátí hodnotu typu Boolean *„false“* V opačném případě navrátí hodnotu *„true“*.

XML soubor sloužící pro import objednávek na straně spřáteleného e-shopu musí mít tvar, jaký lze na obrázku 3.

```
<eshop>
	<order>
		<number>Objednavka 1</number>
		<amount>1000</amount>
		<cookie> cookie uzivatele </cookie>
	</order>
	<order>
		<number> Objednavka 2</number>
		<amount>300</amount>
		<cookie> cookie uzivatele</cookie>
	</order>
</eshop>
```

Obrázek 3: Ukázka struktury XML souboru pro import objednávek

4.2.4 Měsíční přehled

Tato funkce je detailní popis funkce zmíněné v kapitole 4.1.9 v seznamu jako položka d. Pomocí této funkce je každému uživateli poslán měsíční přehled. V tomto přehledu uživatel najde všechny potřebné informace. Mezi tyto informace patří celkový počet kliků neboli přesměrování zákazníků na e-shopy, které uživatel doporučuje libovolnou cestou a jakýmikoliv prostředky. Dále zde vidí počet objednávek, které čekají na zaplacení. Nejsou zde jenom objednávky uskutečněné za poslední měsíc, a to z toho důvodu, aby uživatel měl lepší přehled a zbytečně ho to nemátlo. Jako poslední věc, kterou se pomocí tohoto přehledového emailu dozví, je celková suma provizí za dané objednávky. Tento email je generovaný jednou měsíčně, a to vždy k prvnímu v měsíci.

4.2.5 Zobrazení e-shopů s přehledem, jak si jednotliví uživatelé vedou

Tato funkce je detailní popis funkce zmíněné v kapitole 4.1.8 v seznamu jako položka c. Funkce vrátí data pro sekci přehled pro jednotlivé e-shopy daného uživatele. Zobrazuje data pro každý e-shop, kde zobrazuje, jak si vedou jednotliví uživatelé, kteří tento e-shop doporučují. U každého uživatele doporučující daný e-shop se zobrazuje počet přesměrování zákazníku, počet uskutečněných objednávek a součet jejich cen.

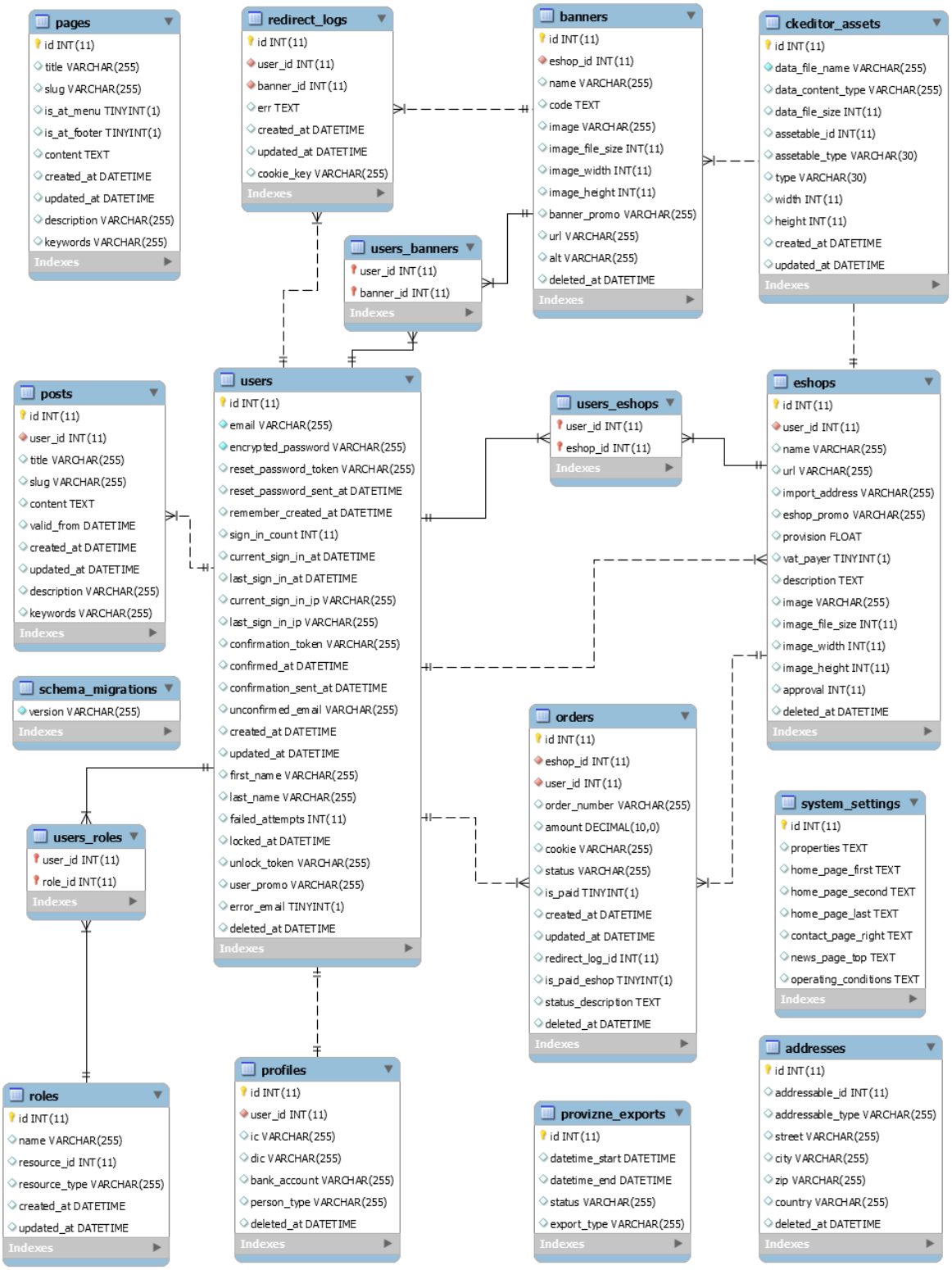
5 Návrh systému

5.1 Technická specifikace

Systém bude postaven na moderním frameworku Ruby on Rails [1], který je postaven nad skriptovacím jazykem Ruby [7]. Tento programovací jazyk byl vybrán z důvodu preferencí firmy. Dále aplikace bude využívat další moderní technologie jako je Javascript [8] a CSS3 [9]. Systém bude komunikovat s databází MySQL [10], která pravděpodobně nepoběží přímo u aplikace, ale bude využita jako cloudová služba. Tento typ databáze byl zvolen také dle preference firmy. Z důvodu očekávaní vysokého počtu dotazů za sekundu a požadavků ze strany firmy, se databáze zvolila jako cloudová služba. Počet dotazů se bude zvyšovat dle počtu partnerů, kdy se pak při plném zatížení očekává desetitisíce až statisíce dotazů za sekundu. Při takové zátěži není možné, aby jí zvládla databáze nasazena pouze na jednom lokálním serveru. Také se očekává vstoupení na zahraniční trhy pro které se hodí více cloudová databáze umožňující replikaci do několika zón. Tato cloudová služba se chová a nastavuje stejně jako vlastní lokální databáze. Jediný rozdíl je ten, že nezajišťujeme chod této databáze, protože ten je zajištován provozovatelem. Dále můžeme jednoduše škálovat její parametry a snadno nastavovat počet replikací.

5.2 Třídní diagram

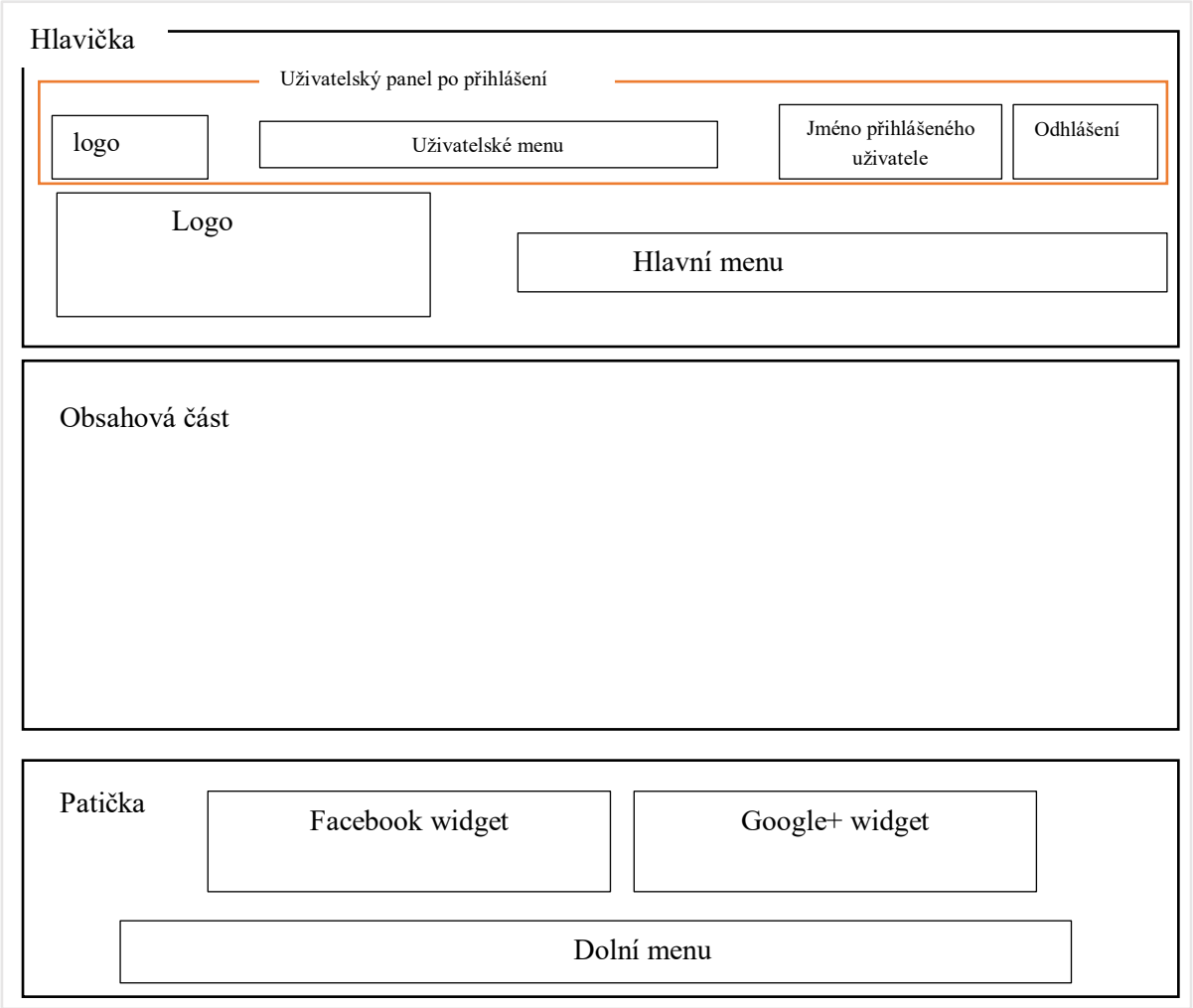
Třídní diagram představuje schéma databáze, které bylo navrhnutu podle potřeb webového provizního systému. V tomto schématu musí být zahrnuta reprezentace jednotlivých uživatelů vlastnící e-shopy, které obsahují reklamní bannery nebo odkazy. Dále je potřeba uchovávat jednotlivé přesměrovaní na dané adresy e-shopů, které musejí být propojené s daným bannerem a uživatelem, díky kterému bylo toto přesměrovaní vytvořeno. Z těchto přesměrovaní mohou na e-shopu vzniknout nějaké objednávky, které jsou následně importovány do tohoto systému, a které je potřeba propojit s e-shopem, ve kterém vznikly, a s uživatelem, díky kterému byla vytvořena. Kromě toho je také potřeba uchovávat další informace jakou jsou uživatelské role, nastavení uživatelů a aplikace nebo měnitelné textové části, kterými jsou například příspěvky na blogu nebo jednotlivé stránky aplikace. Tato struktura lze vidět na obrázku 4.



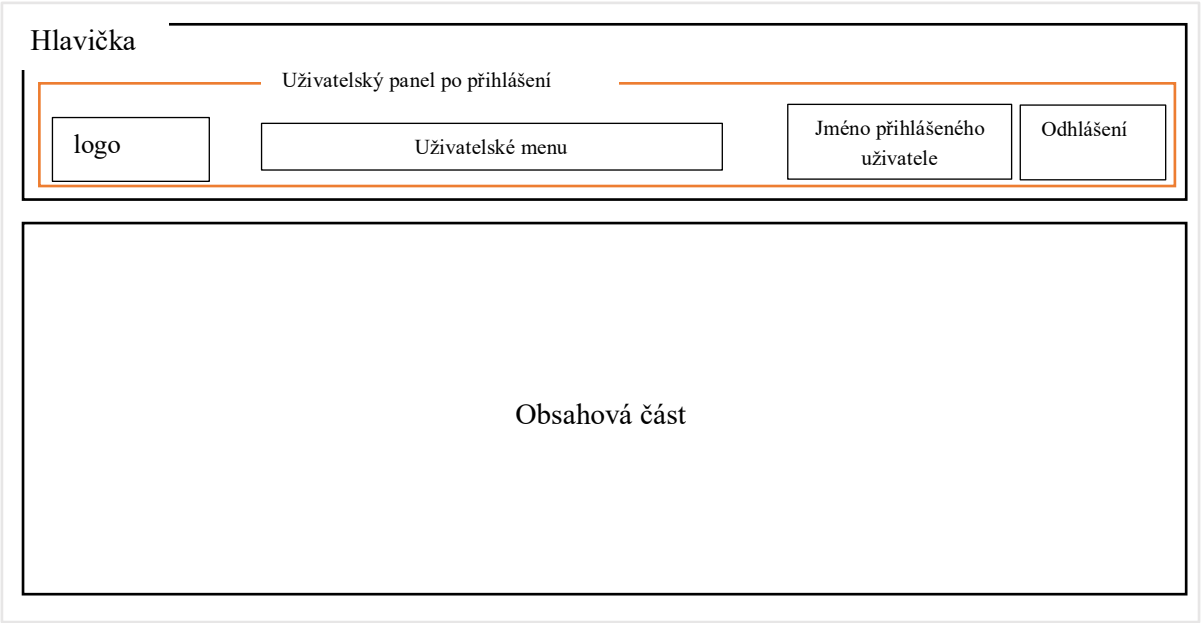
Obrázek 4: Třídní diagram aplikace

5.3 Návrh rozhraní aplikace

Jako každá moderní aplikace, tak i tento webový provizní systém, potřebuje mít nějaké příjemné grafické rozhraní, pomocí kterého, budou jednotlivý uživatelé s tímto systémem komunikovat. Toto rozhraní by mělo být srozumitelné a jednoduše ovladatelné s přehledným rozložením jednotlivých ovládacích prvků. Toto rozhraní musí být optimalizované pro mobilní zařízení. Této optimalizace jsem dosáhl pomocí nejjednodušší cesty, která spočívá ve využití responsivního frameworku. Responsivních frameworků existuje celá řada, ale nejznámější a nejpoužívanější jsou pouze dva. Je to Bootstrap [11] a Zurb Foundation [12]. Pro tuto aplikaci jsem si vybral Zurb Foundation. Jednotlivé návrhy lze vidět na obrázcích 5 a 6.



Obrázek 5: Uživatelské rozhraní pro veřejnou část aplikace

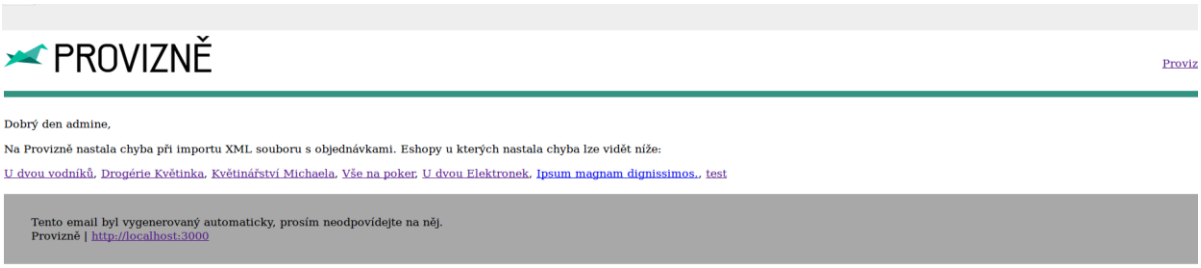


Obrázek 6: Uživatelské rozhraní aplikace pro přihlášeného uživatele

Jak lze vidět z návrhu uživatelského rozhraní na obrázcích 5 a 6, tak veřejná část i část, která potřebuje přihlášení uživatele, jsou téměř totožné. Veřejná část obsahuje navíc v hlavičce logo aplikace a hlavní menu, pomocí kterého lze procházet veřejnou část aplikace. Dále je pak rozdílná o patičku, ve které se nachází propojení na uživatelské stránky od Google+ a Facebooku. V neposlední řadě obsahuje také spodní menu. Po přihlášení uživatele se také zobrazí horní lišta, která je Uživatelský panel, který umožnuje orientaci v části aplikace, která vyžaduje přihlášení. Tento panel obsahuje zmenšené logo, uživatelské menu pro procházení uživatelské části, jméno uživatele a odhlášení. Jméno uživatele je ve tvaru menu, které po najetí zobrazí možnost pro úpravu profilu, nebo změnu hesla. Administrátorská část aplikace je totožná s rozhraním pro přihlášené uživatele, ale navíc obsahuje v uživatelském menu položky umožňující plnou správu aplikace. Tyto položky jsou dostupné pouze pro uživatele s administrátorskými právy. Díky tomuto jednoduchému a sjednocenému rozhraní pro veřejnou a neveřejnou část aplikace je orientace v provizním informačním systému snadná.

5.3.1 Vzhled chybového emailu

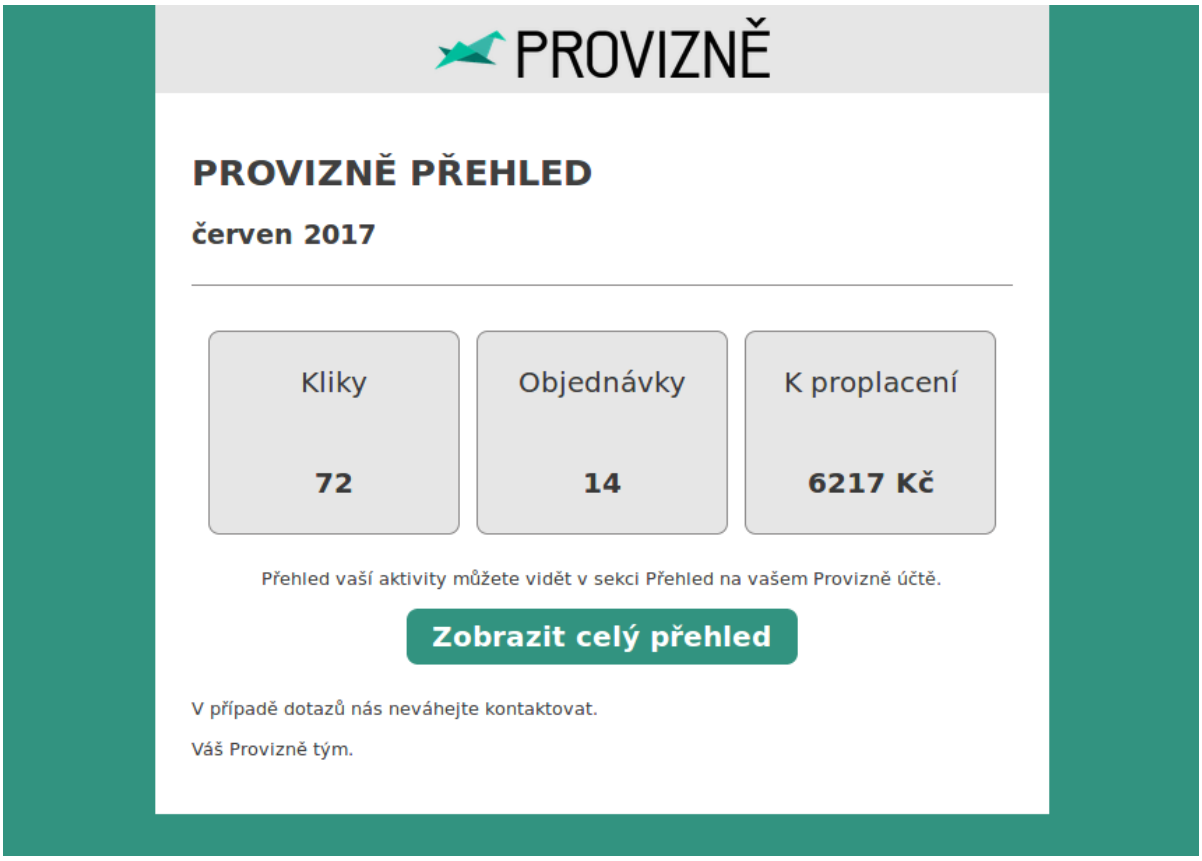
Na obrázku 7 lze vidět návrh vzhledu pro emailové oznámení chyb, které nastanou během importu objednávek z e-shopů. Popis této funkce je v kapitole 4.2.3.



Obrázek 7: Ukázka chybového emailu pro import objednávek

5.3.2 Vzhled emailu pro měsíční přehled

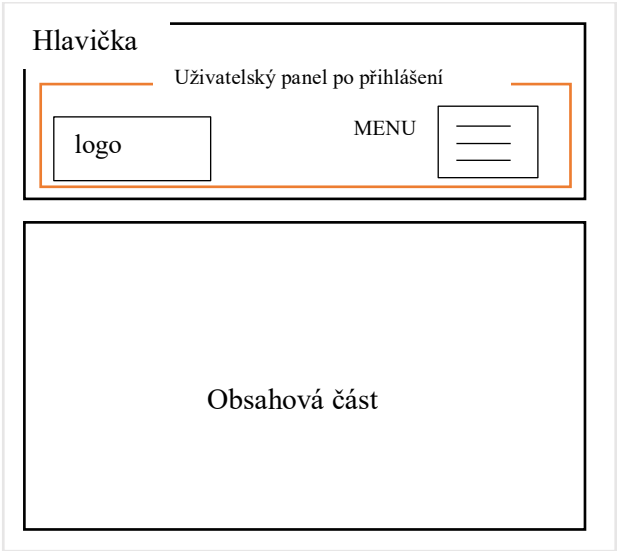
Na obrázku 8 lze vidět návrh vzhledu pro emailový měsíční přehled pro uživatele doporučující e-shopy. Popis této funkce je v kapitole 4.2.4



Obrázek 8: Měsíční emailový přehled

5.3.3 Mobilní UI

Na obrázku 9 lze vidět návrh vzhledu administrátorské části pro mobilní zařízení. Veřejná část je téměř totožná, kde se jenom očekává, že veřejné menu bude pro snadnou orientaci tvořeno pomocí list tlačítka a logo bude přes celou šířku aplikace. Pod logem se pak bude nacházet menu.



Obrázek 9:Návrh mobilního administrátorského UI

6 Rešerše SQL databázových systémů v cloudu

V této kapitole se podíváme na databáze v cloudu, kde se zaměřím na nejznámější a největší poskytovatele této služby na trhu. Než budeme pokračovat, tak si nejdříve objasníme, co vlastně znamená nebo se skrývá pod pojmem databáze v cloudu. Následně se pokusím porovnat jednotlivé nabízené služby databází v cloudu na trhu, které jsem našel, a na které jsem se zaměřil. V závěru této kapitoly vybereme jednu databázovou službu, která pak bude použita v produkčním prostředí dle parametrů a preferencí firmy.

6.1 Co je vlastně databázový systém v cloudu

Databázovým systém v cloudu je myšlen DBS, který je poskytován jako služba přes internet. Tento DBS pak poskytuje dané vlastnosti podle typu použité databázového systému. Některé DBS mají i určitá omezení, která vycházejí z rozhodnutí poskytovatele dané služby z důvodu jeho schopností nebo kvůli bezpečnosti celé služby. Mezi tato omezení například patří maximální velikost databáze, maximální velikost RAM, zakázané určité funkce a další. I přes tato omezení se služba hojně využívá, a to díky převaze výhod oproti nevýhodám. Mezi tyto výhody patří hlavně nižší náklady, protože není nutné pořizovat vlastní hardware. Další je údržba a správa daného systému, kterou nyní řeší poskytovatel dané cloudové databáze. Další výhodou této databáze je možnost škálovatelnosti, díky které můžeme měnit parametry podle potřeb naší aplikace a postupného růstu aplikace i počtu našich zákazníků. Poslední neméně důležitou výhodou je možnost replikace databáze do určitých zón, které umožnuje daný poskytovatel. Nejčastějšími zónami je Evropa, USA a Asie. Replikace databáze znamená vytvoření identických kopií databáze na různých místech. Nejčastěji na různých kontinentech. Poté se provádějí synchronizace všech změn v jednotlivých databázích pro udržení všech kopií identických. Tato replikace se provádí proto, aby daná aplikace byla přístupná na různých místech světa přibližně ve stejném čase nebo jako ochrana proti výpadku celé databáze.

Nyní jsme se již seznámili s pojmem DBS v cloudu, ale neřekli jsme si, jakým způsobem se k němu lze připojit nebo jakým způsobem se za tuto službu platí. Připojení k DBS spočívá v nastavení, které je stejné, jako kdyby se jednalo o DBS lokální. U některých poskytovatelů této služby je potřeba navíc nastavit VPN připojení do jejich sítě z důvodu větší bezpečnosti, ale toto nastavení se liší dle poskytovatele a většinou se připojujeme jako k lokálnímu DBS. Někteří poskytovatelé pak nabízí obě varianty, kde pak záleží pouze na našich preferencích a potřebách. Druh platby za tuto službu se pak také liší dle poskytovatele, ale většinou se u cloudových služeb jedná o cenu za jednu hodinu, kdy při nepřetržitém provozu nabízí nějakou slevu, která je pak výhodnější než hodinová sazba. V neposlední řadě pak většina poskytovatelů poskytuje mnoho variant databáze, díky kterým lze zvolit (škálovat) parametry dle aktuální potřeby naší aplikace, kde se následné náklady na databázi postupně zvyšují s růstem naší aplikace.

6.2 Porovnání SQL DBS na trhu

Na trhu existuje mnoho poskytovatelů SQL DBS v cloudu, proto jsem se v této práci zaměřil na ty nejznámější a podle mě i ty největší poskytovatele na trhu, kterými jsou Google a Amazon. Porovnávat zde budu pouze MySQL [10] DBS, protože tento typ DBS používá tato aplikace, a to hlavně z důvodu preferencí firmy Railsformers.

6.2.1 Google SQL cloud

Google nabízí spoustu služeb, kde jedna z nich je i MySQL databáze. Minulý rok tato databáze nabízela v první generaci omezení na 16 GB RAM a 500 GB prostoru, a nyní již je to 10 TB prostoru a 204 GB RAM. Kromě toho také byla přidaná podpora pro PostgreSQL [13] databázi, která před rokem nebyla ještě vůbec podporovaná. Níže lze vidět jednotlivé vlastnosti databáze, které byly vyčteny z dokumentace. [14; 15]

Vlastnosti

- MySQL DBS verze 5.5 nebo 5.6 v cloudu.
- Pro větší dostupnost jsou všechny údaje replikovány do všech zón.
- Volba možnosti fakturace:
    - Platba pouze za strávený čas potřebný pro přístup k datům.
    - Možnost využití balíčku.
- Celková správa pomocí konzole.
- Jednotlivé instance jsou k dispozici v Evropě, USA a Asii.
- Synchronní nebo asynchronní replikace mezi více zónami s automatickou opravou při chybě
- Importování nebo exportování databáze pomocí mysqldump nebo CSV souboru
- Komptabilita s Javou nebo Pythonem
- Podpora připojení s IPv4 nebo IPv6 adresou
- Podpora zabezpečeného protokolu – SSL
- Automatické zálohy a časové obnovení databáze
- Kompatibilní s normou ISO/IEC 27001

Omezení

- Omezení jednotlivých instancí na velikost 10TB (500 GB 1. generace)
- 208 GB RAM (16 GB 1. generace)
- 25 000 IOPS
- 32 jader procesoru
- Nepodpora uživatelských funkcí
- Následující funkce nejsou podporovány:
    - LOAD DATA INFILE ale LOAD DATA LOCAL INFILE je podporováno.
    - SELECT ... INTO OUTFILE/DUMPFILE
    - INSTALL/UNINSTALL PLUGIN ...
    - CREATE FUNCTION ...
    - LOAD_FILE()
    - Vyšší privilegium uživatele není podporováno
- U MySQL klientského programu není podporovaná funkce mysqlimport bez použití --local možnosti

6.2.2 Amazon SQL cloud

Na rozdíl od Googlu, který nabízí pouze MySQL a PostgreSQL DBS, tak Amazon nabízí nejpoužívanější SQL DBS. Těmito DBS jsou Amazon Aurora [16], Oracle DBS [17], Microsoft SQL Server [18], PostgreSQL, MySQL a MariaDB [19]. Amazon Aurora je typ DBS, která je kompatibilní s MySQL DBS, kde tvrdí, že by měla být až 5x rychlejší. Ovšem tuto informaci nemůžu potvrdit, jelikož

jsem ji nikde neověřoval ani výkonově tyto dva typy neporovnával. Všechny následující vlastnosti byly vyčteny z dokumentace. [20; 21; 22]

Vlastnosti

- Před konfigurované nastavení databáze s možností úpravy
- Monitorování stavu databáze a ceny za využití
- Možnost posílání upozornění o problémech s databází nebo stavu jejího využití
- Možnost použití SSD úložiště
- Automatické zálohovaní a tvorba zálohy aktuálního stavu (tvorba SNAPSHOTŮ)
- Automatická instalace aktualizací databáze
- Možnost replikací
- Možnost šifrovaní

Omezení

- Maximální velikost databáze 6 TB
- Maximální výkon diskových operací může být od 1,000 IOPS do 30,000 IOPS podle použitého typu uložiště.
- 40 jader procesoru
- 244 GB RAM
- Bohužel další informace, kde by byla specifikovaná omezení se mi nepodařilo najít.

6.2.3 SQL DBS s možností vlastní správy

Poslední tady zmíněnou možností je správa vlastního DBS a následné nasazení na nějakou cloudovou službu. Pokud bychom se rozhodli pro tuto možnost, tak zde musíme počítat s práci navíc, kterou tvoří správné nastavení, údržba nebo řešení problémů, které mohou nastat. Opak toho je použití nějakého cloudového DBS, kde tyhle problémy nemusíme řešit, protože se o ně stará poskytovatel cloudového DBS. Ovšem díky použití své vlastní instance rozhodujeme o všech parametrech a omezení, které většinou u cloudových služeb nemůžeme ovlivnit.

6.2.4 Shrnutí

Výběr databáze záleží na konkrétních požadavcích a preferencí provozovatele této aplikace. Pro testování této aplikace jsem použil vlastní nasazený MySQL DBS pomocí Dockeru [23]. Rozhodnutí použití vlastního DBS bylo z důvodu, že cloudová verze je poskytována jako placená služba a já jsem ji neměl předplacenou. Dalším důvodem také bylo to, že díky použití Dockeru, si tuto aplikace může každý jednoduše vyzkoušet i bez potřeby mít předplacenou cloudovou verzi DBS. Cloudová služba DBS je určena hlavně pro produkční prostředí.

7 Implementace a struktura testovacího nasazení

V této kapitole bych chtěl nejprve popsat a shrnout průběh implementace tohoto webového provizního informačního systému a pak bude pokračovat popis ohledně struktury testovacího nasazení.

7.1 Implementace

Tento systém jsem implementoval postupně po malých funkčních celkách, které postupně rozšiřovali funkcionalitu tohoto systému. Tyto části se pak většinou hned testovaly a pokud byly nalezeny nějaké chyby, tak jsem je mohl hned opravit. Výhodou takového vývoje je to, že vždy po dokončení nějaké části, máme funkční stav, který lze testovat.

Samotnou implementaci jsem začal vytvoření základní aplikace, která umožnovala přihlašování uživatele. Následně jsem pracoval na veřejné statické části aplikace, kde jsem vytvořil celkový vzhled, hlavní stránku a kontaktní stránku s kontaktním formulářem. Poté jsem přidal podporu uživatelských rolí. Následně jsem přidal podporu pro vytváření nových stránek s možností nastavení zobrazení této stránky v horním a spodním menu. Implementací veřejné části jsem dokončil přidáním podpory jednoduchého blogu, na kterém lze přidávat články.

Následně jsem pokračoval přidáním hlavní funkčnosti systému, kde jsem nejprve postupně přidal podporu pro e-shopy, bannery, objednávky a vytvořil jsem pro ně plnou administraci s korektními právy. Poté jsem připravil podporu pro udržování záznamů o přesměrování na e-shopy, které také udržují hash kód, který pak obsahuje objednávka z e-shopu, která byla vytvořena přes tento systém. Díky tomuto kódu lze pak identifikovat uživatele, který tuto objednávku doporučil a následně mu pak připsat provizi, kdy schválení připsání této provize řeší administrátoři systému. Následně jsem mohl vytvořit funkcionalitu pro přesměrování jednotlivých doporučených uživatelů na konkrétní e-shop se zaznamenáním potřebných hodnot pro identifikaci uživatele, který ho doporučil. Pak jsem ještě vytvořil funkci importující objednávky z e-shopu, které byly vytvořeny přes tento systém, s korektním spárováním uživatele na straně systému. Nakonec jsem vytvořil různé přehledy, mailové obeznámení, funkcionální testy a další.

Během implementace jsem se nesetkal s žádným problémem, který bych nakonec nevyřešil sám nebo s radou od programátorů z firmy.

7.2 Struktura testovacího nasazení

Aplikace je navržena tak, aby bylo možné jí jednoduše nasadit do cloudového řešení, díky kterému bude schopna zvládat obrovské zatížení. Tento návrh je řešen pomocí Dockeru [23], díky kterému lze aplikaci nasadit téměř kdekoliv, a také nabízí jednoduchou škálovatelnost. Aplikace je závislá na MySQL databázi, ale pokud by se udělala menší úprava, tak ji lze použít s jakoukoliv SQL databází. Testovací hardware

Pro testovací účely jsem dostal přístup ke dvěma školním serverům. Na jednom serveru byl nainstalovaný SUSE Linux Enterprise Server 12 a na druhém byl Windows server 2016 Datacenter. Na linuxovém stroji běžela aplikace a Mysql DBS pod Dockerem a na windowsovském stroji pak běžel generátor vytvářející zátěž pro aplikaci. MySQL DBS během testování neběžel v cloudu z důvody, že se jedná o placenou službu a proto byla také nasazena pomocí Dockeru.

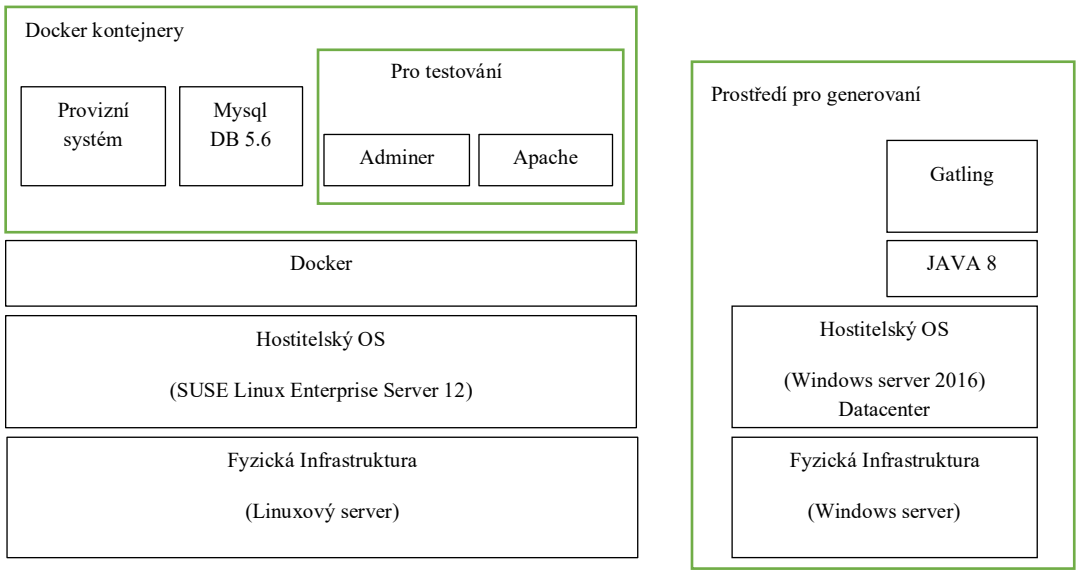
Linuxový server konfigurace:

Procesor:	Intel® Xeon(R) CPU E5-4610 0 @ 2.40GHz × 18
Paměť:	977,8 GB
Grafická karta:	Gallium 0.4 on llvmpipe (LLVM 3.4, 256 bits)
Disk:	3,7 TB

Windows server konfigurace:

Procesor:	2x Intel® Xeon(R) CPU E5-2690 0 @ 2.90GHz × 9
Paměť:	384 GB
Grafická karta:	nezjištěno
Disk:	1,63 TB

7.2.1 Struktura aplikace



Obrázek 10: Struktura nasazení aplikace

Ze struktury na obrázku 10 můžeme vidět kromě konkrétního rozdělení samotné aplikace a zátěžové aplikace na dva stroje to, že kromě aplikace a databáze v Dockeru, ještě používám Adminer [24] a Apache [25]. Adminer slouží pro jednodušší správu MySQL databáze. Samotný webový server Apache je zde nutný kvůli simulaci webové adresy jednotlivých e-shopů, na kterou mají být přesměrovaní potencionální zákazníci, získaní přes tuto službu.

8 Nástroje pro optimalizaci a návrh generátoru zatížení

Předtím, než jsem začal cokoliv optimalizovat, tak jsem se nejprve snažil najít co nejvíce informací a možností s řešením takového tématu. Během tohoto hledání jsem se hlavně snažil seznámit s technikami, jenž jsou použity pro optimalizaci. Dále s jejich nástroji, které by mi usnadnily práci a pomohly mi vyhledat místa, pro která je potřeba se pokusit o nějakou optimalizaci. Po mnoha hodinách hledání jsem zjistil, že pro optimalizaci existují dva typy nástrojů. První je určen pro odhalení počtů požadavků do databáze pro jednotlivé části a funkce aplikace. Dále pomáhá zjistit dobu vykonání jednotlivých operací a dobu zobrazení. Druhý typ nástrojů pak slouží pro vytížení aplikace většinou obrovským množstvím webových požadavků, které simulují reálné uživatele, kteří přistupují do aplikace a tím pomáhají identifikovat slabá místa a také zhodnotit optimalizaci na obrovské zátěži. Tyto nástroje budou popsány v následujících dvou podkapitolách.

8.1 Nalezení vhodného nástroje pro profilování aplikace

Před začátkem jakékoli optimalizace systému je nutné nejprve vybrat vhodný nástroj, který nám umožnuje profilovat aplikaci, díky které jsme schopni například vidět, jak naše aplikace vytěžuje databázi, jaká doba je potřeba pro vykonávání určitých operací nebo jak dlouho trvá zobrazení aplikace. Během vybíraní tohoto nástroje jsem narazil na několik aplikací, které to umožnují a které zde budou popsány. Všechny zde nalezené nástroje lze jednoduše přidat pomocí gemu.

8.1.1 ruby-prof

Tento nástroj nabízí měření několika typů hodnot jako je procesový čas, čas CPU, paměť a další. Výsledky měření jsou vytvářeny pro každý webový požadavek nebo pro určitý blok a ukládají se na určené místo. Z těchto výsledků si lze pak vygenerovat čitelný výstup do několika formátů jako je HTML graf, textový výstup, textový graf a další. Z těchto výstupů pak lze vyčíst požadované informace. Na začátku se některé z těchto typů výstupů mohou zdát trošku nepřehledné, ovšem po chvilce soustředění se v nich dá celkem jednoduše orientovat. Více informací lze najít na oficiálních stránkách. [26]

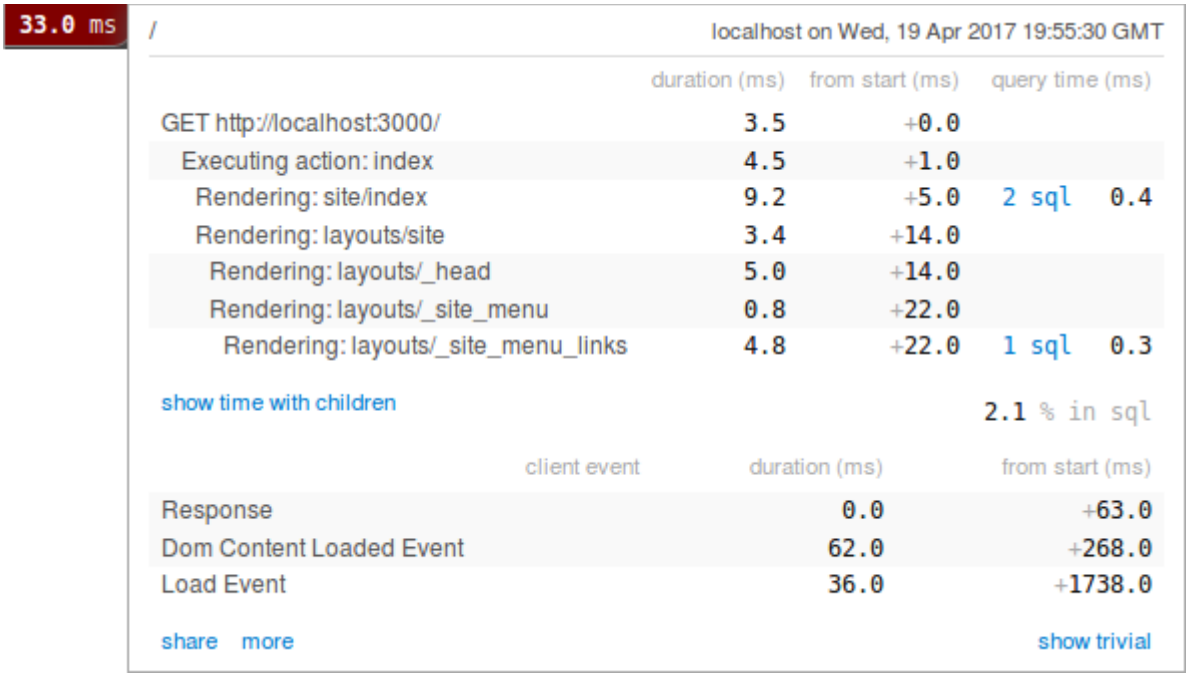
Tento nástroj jsem zkoušel několikrát podle různých návodů, ovšem ani z jednoho se mi nepodařilo dostat do plného funkčního stavu. [26; 27] Místo toho jsem pokaždé jenom obdržel nějakou chybu, u které se mi bohužel nepodařilo přijít na to, jak ji opravit. Celkem mě to mrzí, jelikož mi tento nástroj přišel hodně zajímavý a nabízel různé typy měření. Tento nástroj by možná šel použít také pro profilování během výkonového testování, díky kterému, bychom mohli získávat dva typy výsledků najednou, které by mohli usnadnit identifikaci problému.

8.1.2 Rack-miniprofiler

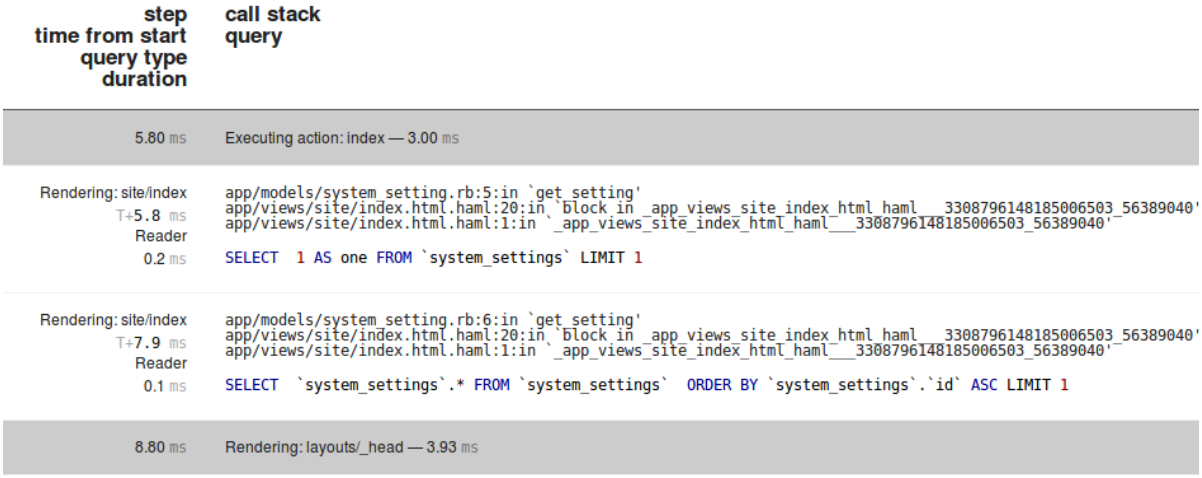
Rack-miniprofiler je velmi jednoduchý a nabízí všechny potřebné funkce, které jsou potřeba pro profilováni aplikace. Mezi tyto podporované funkce patří profilování využití databáze, která může být typu MySQL [10], PostgreSQL [13], Oracle databáze [17] nebo MongoDB [28]. Mezi další funkci patří profilovaní zásobníku, díky kterému jsme schopni vidět čas strávený v určitém rozšíření (gemu). Tento výsledek je pak zobrazen pomocí grafického výstupu, o který se stará rozšíření – gem flamegraph [29], bez kterého by tato funkce nebyla k dispozici. Poslední nabízenou funkcí je podpora profilovaní paměti RAM, která nabízí zobrazení použité paměti RAM pouze pro daný požadavek. Dále zobrazuje informace o GC [30] statistikách, které jsou informace získané z GC. Kromě toho, taky samozřejmě zobrazuje globální alokované metriky, mezi které patří například celková pamět nebo počet objektů.

Aby tento nástroj správně fungoval a mohl zobrazovat všechny tyto typy informací, tak je potřeba přidat gemy, na kterých je tento nástroj závislý. Tyto potřebné gemy jsou flamegraph, stackprof [31] a memory_profiler [32]. Flamegraph vytváří grafický výstup výsledků, Stackprof pak slouží pro profilování zásobníku a memory_profiler nám slouží pro profilovaní paměti. Každý z těchto gemů bychom mohli sice použit zvlášť, ale díky tomuto nástroji máme tyto všechny výsledky na jednom místě s jednotným přístupem a s přidanou funkcionalitou navíc, kterou přidává tento gem. Dokonce tento nástroj podporuje kromě jazyka ruby také profilování .NET aplikace, pro kterou má vlastní verzi [33].

Následné profilovaní aplikace s tímto nástrojem je pak velice jednoduché, jelikož pokud máme podle dokumentace všechno správně nastavené, tak pak stačí pouze navštívit část webu, kterou chceme profilovat. Výsledkem je defaultní zobrazení malé ikonky v levém horním rohu webové stránky. Tato ikonka zobrazuje celkový čas, který byl potřeba pro zpracování a zobrazení této stránky. Jestliže na tuto ikonku klikneme, tak se nám zobrazí základní údaje o tomto zpracování. Tyto údaje, jak ukazuje obrázek 11, jsou jednotlivé doby trvání pro zpracování a zobrazení, počet SQL dotazů, čas jejich vykonání a další údaje. Jestliže pak klikneme na jakýkoli SQL dotaz, tak získáme více informací o těchto provedených SQL dotazech. Tento výsledek lze vidět na obrázku 12.



Obrázek 11: Rack Mini Profiler – detail



Obrázek 12: Rack Mini Profiler – SQL detail

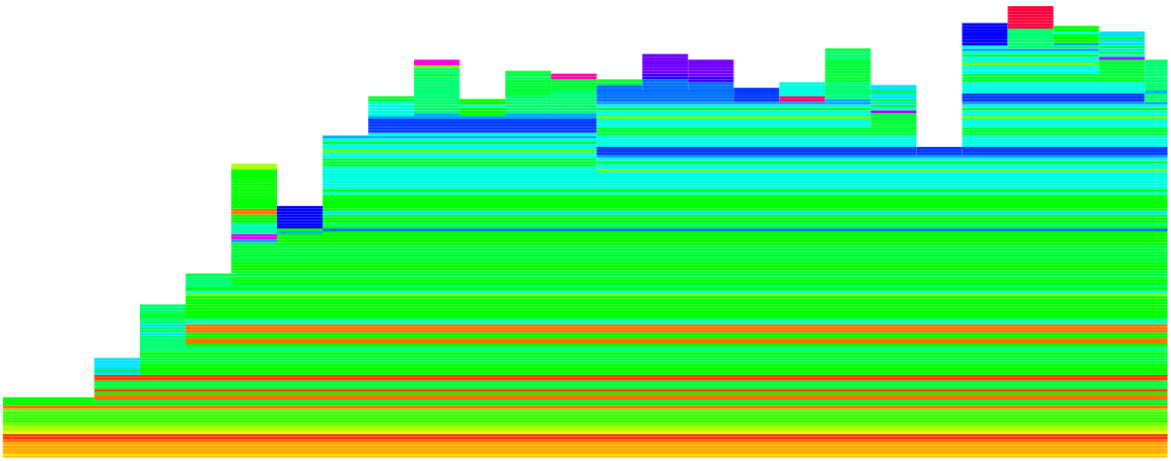
Jestliže chceme profilovat využití paměti RAM nebo vidět hodnoty z Garbage kolektoru [30], tak stačí pouze přidat parametr určující typ výpisu za URL adresu, díky kterému po odeslání dostaneme výpis všech podporovaných hodnot. Pro zobrazení dat pro profilovaní paměti RAM přidáme tento parametr: „?pp=profile-memory“ k naší profilované webové adrese a po odeslání tohoto požadavku získáme data pro profilovaní. Velmi malá část výstupu lze vidět v ukázce, která se nachází na obrázku 13). V této ukázce pak můžeme vidět celkovou velikost alokované paměti. Také zde lze vidět kolik paměti je alokováno jednotlivými gemy a kolik mají vytvořených objektů. Díky tomuto jsme schopni jednoduše optimalizovat využívání paměti.

```
Total allocated: 658152 bytes (8093 objects)
Total retained:  32887 bytes (166 objects)

allocated memory by gem
-----------------------------------
   138600  activesupport-4.2.6
    92043  activerecord-4.2.6
    84730  actionview-4.2.6
    79379  actionpack-4.2.6
    64153  haml-4.0.7
...
allocated objects by gem
-----------------------------------
     1536  activerecord-4.2.6
     1229  activesupport-4.2.6
     1140  actionview-4.2.6
     1076  haml-4.0.7
      877  actionpack-4.2.6
      485  provizne/app
      356  nokogiri-1.7.1
      304  arel-6.0.4
      240  ruby-2.2.1/lib
```

Obrázek 13: Ukázka výstupu z nástroje Rack Mini Profiler pro profilování využití paměti RAM

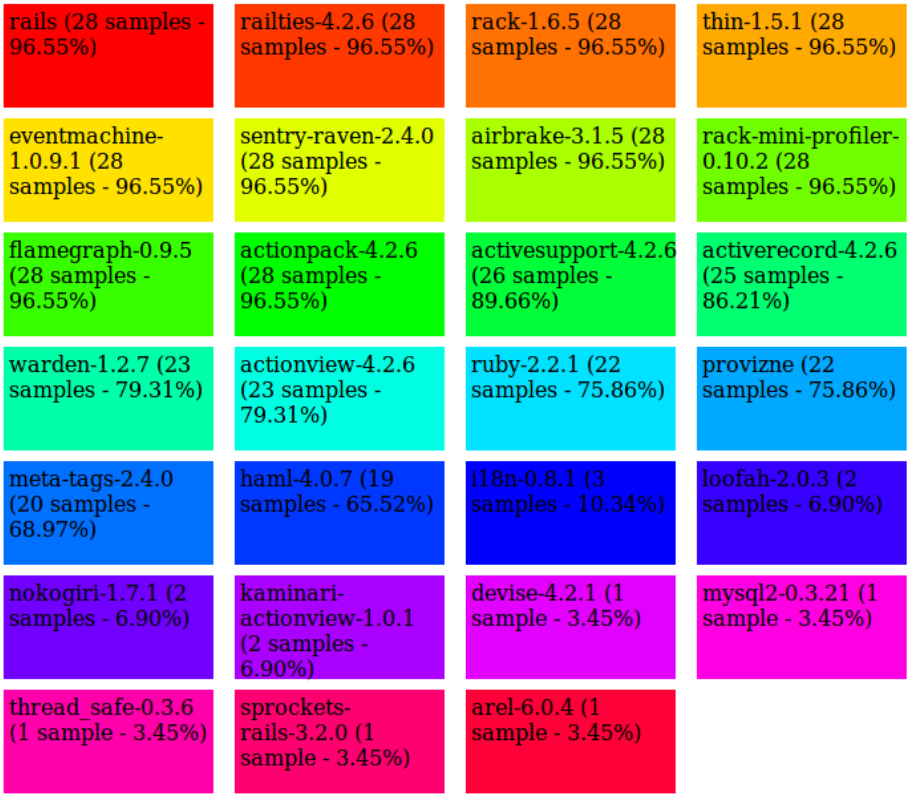
V další ukázce je obsažen grafický výstup pro profilovaní pomocí zásobníku, který lze vidět na obrázku 14 a který byl vytvořen z hlavní veřejné stránky pomoci gemu Flamegraph. Pro získání tohoto

typu výstupu je potřeba přidat do našeho webového požadavku tento text: „?pp=flamegraph“, který následně po odeslání vytvoří tento grafický výstup s jeho legendou. Výška grafu indikuje hloubkou zanoření v zásobníku, kde výškou grafu mám na mysli osu Y. Na ose X je pak čas. V grafu pak lze přiblížit nebo oddálit konkrétní místo pomocí myši.



Obrázek 14: Rack Mini Profiler – Grafický výstup

Ve spodní části výpisu nalezneme legendu, nacházející se na obrázku 15), která objasňuje, co každá barva znamená. V poznámce se pak nachází procentuální vyjádření času, který daný požadavek strávil uvnitř daného rámce obsaženého v zásobníku. Například tato aplikace se jmenuje „provizne“. Zdá se, že jsme v samotné naší aplikaci strávili 75,86 % času, kde tento výsledek není nijak špatný.



Obrázek 15: Rack Mini Profiler – Legenda ke grafickému výstupu

Shrnutí:

Jak lze vidět, tak tento nástroj nabízí podporu při optimalizaci díky zobrazení množství různých typů dat. Z tohoto důvodu jsem se rozhodl si tento nástroj vybrat pro pomoc během optimalizace.

8.2 Nalezení vhodného nástroje pro vytížení systému

Při optimalizaci systému kromě nástroje pro profilování aplikace, který byl popsán v minulé podkapitole, potřebujeme také vidět chování aplikace při jejím plném zatížení, díky které můžeme identifikovat další úzká hrdla systémů, které většinou nelze objevit při nízkém zatížení během profilování.

Během hledání jsem narazil na několik nástrojů, které řeší tuto problematiku. Většina těchto nástrojů provádí vytížení systému pomocí simulace reálných uživatelů, kteří následně vytváří požadavky na danou aplikaci, díky kterým ho plně vytěžují. Některé méně propracované aplikace nesimulují reálného uživatele, ale přímo generují obrovské množství požadavků na systém, díky kterému je pak simulace méně podobná reálnému vytížení systému. Také každá aplikace řeší simulaci uživatele svým způsobem, a to pomocí vláken, procesů nebo kombinují obě možnosti. Některé aplikace vytvářejí simulaci uživatele, která se blíží reálnému uživateli více a jiné tuto simulaci zvládají méně. Já jsem se snažil vybrat nástroj, který se bude reálnému uživateli co nejvíce přibližovat. Nalezené nástroje budou popsány níže.

Pro označení místa v kódu a následné odkazovaní v ukázkách používám tuto syntaxi například (řádek 1), který znamená řádek v kódu označený (1).

8.2.1 RSPEC

Jako první jsem zkoušel zjistit, zda použitý framework Ruby on Rails má nějakou přímou podporu, která by optimalizaci řešila nebo usnadnila podobně jako použitý testovací framework RSPEC, který jsem zde použil pro funkční testování aplikace. RSPEC sice nabízí také výkonové testování, ale přišlo mi, že toto řešení je spíše určeno pro testovaní jenom určitých části aplikace jako je přihlášení nebo vytvoření třeba stránky než testování aplikace jako celku, který jsem chtěl zde testovat. Proto jsem se rozhodl, že tento nástroj nevyužiji a hledal jsem dále. Podrobné informace lze najít na oficiální stránce tohoto nástroje [34].

8.2.2 `Rails-perftest`

Jakmile jsem framework RSPEC zamítl, tak jsem následně narazil na tento nástroj Rails-perftest [35], který šel přímo integrovat s projektem v Ruby on Rails. Psaní testů bylo podobné jako v případě klasických testů ve frameworku RSPEC, kde jednotlivé testy vypadaly velice jednoduše a přehledně. Instalace byla jednoduchá díky použití gemu. Ovšem jakmile jsem tento nástroj zkoušel, tak bohužel se mi ho nepodařilo správně nastavit. Stále jsem dostával nějakou chybu. Proto jsem se pro tento nástroj nerozhodl. Kdyby se mi ovšem tento nástroj podařilo správně nastavit, aby fungoval, tak bych se pro něj nakonec stejně nerozhodl z důvodu jeho jednoduchosti pro jednotlivé testy, u kterých mi přišlo, že v nich nelze psát složitější testovací případy.

Podle mého názoru se tento nástroj hodí pouze pro jednoduché výkonové testování, kde je potřeba pouze vytížit nějakou část aplikace, místo zatížení aplikace jako celku.

8.2.3 Locust

Jako další jsem našel Locust.io [36], který je moderní opensource testovací framework, jenž je napsaný v programovacím jazyku python [37]. Tento nástroj se mi líbil, jelikož psaní testů vypadalo jednoduše s možností rychlého psaní testů. Kromě tohoto mě také oslnila podpora spuštění instancí na více strojích, díky kterým by měl umět simulovat miliony uživatelů. Tyto funkce mě oslovily, a proto jsem se rozhodl, že ho vyzkouším. Pro vyzkoušení jsem použil několik jednoduchých testů pro pár stránek a také jednoduché přihlášení uživatele. Spouštění více instancí na více strojích jsem se rozhodl prozatím nezkoušet. Po vyzkoušení jsem zjistil, že psaní testů je opravdu jednoduché. V ukázce testu, zobrazeném na obrázku 16, lze vidět jednoduchý test pro testovaní stránky dashboard.

```
from locust import HttpLocust, TaskSet, task

class UserBehavior(TaskSet): # (1)
    def on_start(self): # (2)
        # on_start je spustěna předtím, než jsou spuštěné úkoly (@task)
        self.login()

    def login(self): # (3)
        self.client.post("/users/sign_in",
                         "user[email]=radek.turek@example.cz
                         &user[password]=Passw0rd") # (4)

    @task(1)  # (5)
    def index(self): # (6)
        self.client.get("/dashboard")

class LoginedUser(HttpLocust): # (7)
    task_set = UserBehavior # (8)
    min_wait = 5000
    max_wait = 9000
```

Obrázek 16: Ukázka testu pro nástroj Lucust

V této ukázce kódu lze vidět dvě třídy UserBehavior a LoginedUser. Třída UserBehavior (viz řádek označený 1) dědí z třídy TaskSet a obsahuje celou logiku jedné skupiny testů, kde každý test je definován pomocí anotace @task (řádek 5). Pomocí jeho parametru můžeme definovat četnost spouštění tohoto testu. V tomto příklade je definován pouze jeden test se jménem index (řádek 6), který přistupuje na stránku dashboard. Před zpracováním skupiny jednotlivých testů v dané třídě, lze definovat nějaké akce, které je potřeba provést před začátkem zpracování jednotlivých testů, a to pomocí definované funkce on start (řádek 2). Toto je z důvodu, abychom mohli před vykonáváním jednotlivých testů provést například nějaké nastavení, které bude platné pro všechny testy v dané třídě, nebo například provést přihlášení uživatele. V této ukázce lze vidět, že tato funkce je využita pro přihlášení uživatele, které je nezbytné pro přístup na stránku dashboard. Toto přihlášení je realizováno pomocí funkce login (řádek 3), která pošle POST požadavek obsahující data pro přihlášení uživatele, která jsou v tomto případě email a heslo uživatele. Tento požadavek je následně vytvořen pomocí třídy Client, která je součástí testovacího frameworku, a její funkce post(), která očekává dva parametry. Prvním parametrem je adresa, na kterou se má tento požadavek vytvořit. Druhým parametrem jsou pak data, která má tento POST požadavek obsahovat (řádek 4).

Třída LoginedUser dědí z třídy HttpLocust a obsahuje nastavení pro celý test. V našem příkladu vidíme nastavení naší třídy UserBehavior do proměnné task_set, která obsahuje třídu, která je použita

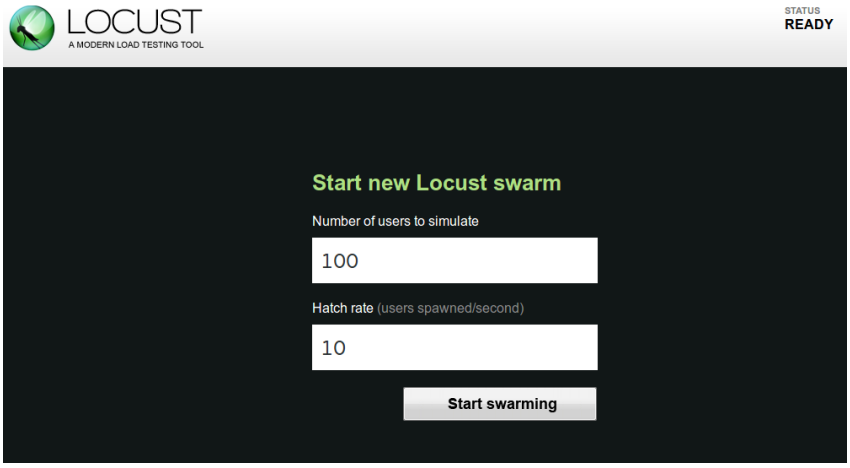
pro celkové testováni. Dále lze vidět ještě nastavení minimálního a maximálního časování pro uživatele, které určuje dobu mezi provedením každého požadavku.

Jakmile máme vytvořený nějaký testovací případ, v tomto případě přístup přihlášeného uživatele na stránku dashboard, tak můžeme tento test spustit. Spuštění samotného testu je potřeba udělat v několika krocích. Pokud se nacházíme v hlavním adresáři projektu, tak nejprve musíme spustit locust webový monitor, který slouží pro ovládaní celého testu a zpracování výsledků. Monitor spustíme pomocí tohoto příkazu:

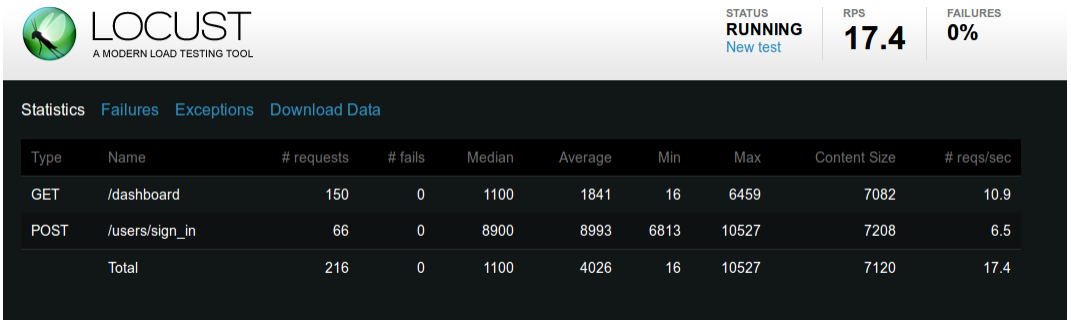
```
locust -f test/locust/locustfile.py --host=http://localhost:3000
```

Kdybychom se nacházeli přímo ve složce, kde se nachází náš test, který chceme spustit, a název souboru by byl locustfile.py, tak bychom mohli vynechat z příkazu pro spuštění přepínač -f s cestou k danému souboru, ve kterém je definován průběh testu.

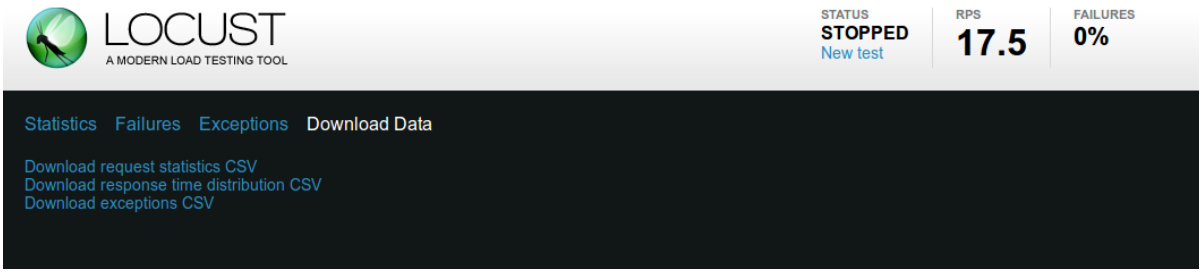
Jakmile máme spuštěný webový monitor, tak je potřeba vstoupit do jeho webového rozhraní, které naslouchá na defaultním portu 8089, a nejdříve nastavit celkový počet uživatelů a také počet uživatelů, kteří budou přidáni každou sekundu. Poté klikneme na tlačítko „Start swarming“, které aktivuje samotný průběh testu. Nastavení pro uživatele lze vidět na obrázku 17. Průběh samostatného testu lze vidět na dalším obrázku 18, na kterém najdeme aktuální počet aktivních uživatelů a celkový počet požadavků za sekundu. Poslední obrázek 19 je pak ukázkou pro export výsledků.



Obrázek 17: Lucust-nastavení uživatelů



Obrázek 18: Locust-Průběh testu



Obrázek 19: Locust-Získání výsledků

Pro tento nástroj jsem se nakonec nerozhodl z toho důvodu, že není stavěný pro vytváření složitější simulace uživatele, který by procházel aplikaci přes více stránek a simuloval tak reálného uživatele. Dalším důvodem byla absence výstupních grafů a také občasný reset výsledků, které pak následně začínaly od nuly. Dále zde není jednoduchá možnost ověření získané stránky v testech, která má za následek úspěšné procházení testů i v případech dostání jiné stránky, než kterou jsme očekávali. Posledním mínusem, který tento nástroj má, je nastavení uživatelů a následné spuštění testování přes webové rozhraní bez časového limitu, a také nutnost ručního uložení výsledků, které ztěžuje spuštění testů a následné zpracování jejich výsledků programově.

Podle mého názoru se tento nástroj hodí pro jednoduché a rychlé vytížení, ale jakmile potřebujeme složitější simulaci zatížení a automatické spouštění testů, tak v tomto případě bychom měli zvolit jiný nástroj, kterým může být například JMeter [38] nebo Gatling [39], které jsou popsány níže.

8.2.4 JMeter

Tento nástroj je hojně využívaný, nabízí mnoho funkcí a je napsaný v Jave, kde jednotliví uživatelé jsou simulovány ve vláknech. Pomocí tohoto nástroje můžeme testovat různé typy aplikací, serverů a protokolů, které lze vidět níže:

- Web-HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, …)
- SOAP / REST webové služby
- FTP
- Databáze přes JDBC
- LDAP
- Mail-SMTP(S), POP3(S) and IMAP(S)
- Nativní příkazy nebo shell skripty
- TCP
- Java objekty

Přímou zkušenost s tímto nástrojem bohužel nemohu popsat, jelikož i když mě hodně zaujal, tak jsem se rozhodl ho nezvolit, protože mě více oslovil nástroj Gatling, protože nabízí vytváření testů pomocí kódu.

8.2.5 Gatling

Poslední nástroj, který jsem našel, byl Gatling, který je hodně využívaný a má dobrou komunitou. Sice tento nástroj nabízí pouze testování webových aplikací na rozdíl od JMeteru, ale tato absence možnosti testování ostatních typů služeb mi v tomto případě nevadila. Tento nástroj je z většiny napsaný pomocí jazyka Scala [40] a jednotliví uživatelé jsou simulováni pomocí vláken. Jeho plusem

je podle mě přehlednější dokumentace a výsledky z jednotlivých testů, které jsou jako HTML výstupy, stejně jako v JMeteru, ale s pěknějším vzhledem a grafy. Posledním plusem je možnost programování testů, které zde vypadají mnohem přehledněji. Tyto vlastnosti mě přesvědčili k tomu, abych si tento nástroj vyzkoušel a následně jsem si ho vybral pro výkonové testování.

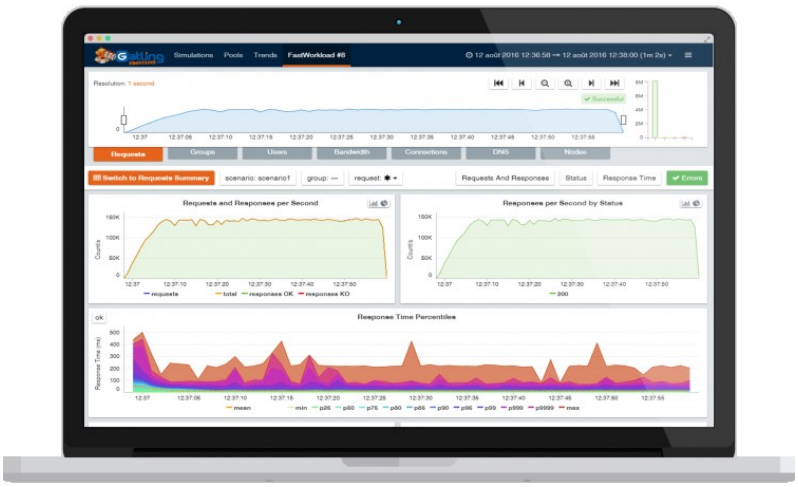
Jednotlivé testy se zde programují v jazyce Scala, který vychází z Javy, ze které plně podporuje její knihovny. Tento jazyk se snaží integrovat rysy objektově orientovaného programování společně s funkcionálním přístupem.

```
import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration.

class BasicSimulation extends Simulation { // 1
  val httpConf = http
    .baseURL("http://localhost:3000") // 2
    .userAgentHeader("Mozilla/5.0 (Windows NT 5.1; rv:31.0)
                      Gecko/20100101 Firefox/31.0") // 3

  val scn = scenario("BasicSimulation") // 4
    .exec(http("req_get_homepage") // 5
    .get("/")) // 6
    .pause(5) // 7

  setUp( // 8
    scn.inject(atOnceUsers(1)) // 9
  ).protocols(httpConf) // 10
}
```

Obrázek 20: Ukázka testu pro nástroj Gatling

V této ukázce na obrázku 20 je jednoduchý test, který přistupuje na hlavní stránku aplikace. Tato ukázka je hodně podobná ukázce, která se nachází v oficiální dokumentaci. V Gatlingu je celkový test psaný ve třídě, která musí dědit ze třídy Simulation (viz řádek označený 1). V této třídě se pak nachází funkce setUp (řádek 8), která obsahuje jednotlivé testové případy. Každý tento případ je pak definován pomocí scény (řádek 4), která obsahuje průběh jednotlivého testovacího případu. V této scéně nejdříve nastavíme název požadavku (řádek 5) pomocí funkce exec. Následně nastavíme, že tento požadavek bude typu GET, a kam bude směřován pomocí funkce get (řádek 6). Nakonec počkáme 5 sekund, které simulují čtení stránky nebo rozhodování uživatele (řádek 7). Pro každou scénu pak lze ještě nastavit další parametry. V této ukázce se nastavuje pouze počet uživatelů pro tuto scénu, kde máme nastavenou simulaci pouze pro jednoho uživatele (řádek 9). Kromě těchto nastavení je potřeba nastavit ještě globální nastavení pro všechny testové případy, které nastavíme pomocí funkce protocols v setUp metodě (řádek 10). V tomto nastavení by měla být minimálně základní adresa, která obsahuje adresu testované aplikace (řádek 2). V této ukázce ještě definujeme nastavení pro typ prohlížeče, který bude simulován (řádek 3). Jak lze vidět z této ukázky, tak vytvoření testovacího případu je velice snadné a přehledné díky kterému lze se v testech jednoduše orientovat. Na obrázcích 21 a 22 se nachází ukázka testového výstupu a jeho grafů. Tyto obrázky byly převzaty z dokumentace.

Obrázek 21: Gatling-Ukázka výstupu testu



Obrázek 22: Gatling-Ukázka grafů

8.3 Generátor zatížení systému

Jakmile jsem vybral testovací nastroj pro zátěžové testování, v mém případě Gatling, tak jsem mohl začít s vytvářením testů, které mi budou vytvářet zatížení aplikace. Toto zatížení je potřeba proto, abych zjistil, jak se daná aplikace chová při daném zatížení a pokusil se objevit její slabá místa a omezení. Dále je tento generátor vytížení systému také výhodný pro možnost opakovaného vytížení, díky kterému pak následně můžeme porovnávat jeho jednotlivé výsledky pro optimalizovanou verzi s neoptimalizovanou.

8.3.1 Návrh generátoru a rozložení vytížení

Při návrhu a implementaci generátoru jsem se snažil klást důraz na to, aby výstupní vygenerované zatížení bylo co nejvíce podobné reálnému zatížení systému, které by vytvářeli běžní uživatelé využívající tuto službu.

Sice jsem neměl k dispozici reálné výstupy se statistikami přístupů z této služby, z důvodu spuštění této aplikace prozatím pouze v testovacím režimu na adrese http://provizne.cz, ale díky určenému typu použití a známe funkčnosti, jsem mohl určit toto vytížení. Díky tomu jsem mohl navrhnout a vytvořit tento generátor zatížení.

Předtím než jsem se pustil do vytváření testů pro vytváření zátěže, tak jsem si nejprve promyslel, jak tato zátěž bude vypadat. Výsledkem bylo rozdělení celkového zatížení na několik menších částí, kde každá část bude představovat vytížení aplikace jiným druhem uživatelů. V každé této skupině budou uživatele představovat a mít stejné chování, které je předpokládané pro využití dané části aplikace. Zatížení je hlavně rozděleno na tyto skupiny:

1. Uživatelé pro přesměrovaní na službu partnera.
2. Registrované a přihlášené uživatele.
3. Neregistrované uživatele.

Uživatele pro přesměrovaní na službu partnera

První skupinou jsou uživatelé, kteří využívají službu nepřímo, a to tak, že pouze kliknou na nějaký odkaz, který je následně přesměruje na službu partnera. Ovšem toto přesměrování je až konečným výsledkem, jelikož uživatel je nejdříve přesměrován na náš provizní sytém, který tento požadavek zpracuje a až pak ho následně přesměruje na službu našeho partnera. Těchto požadavků je předpokládáno obrovské množství a mělo by být i největším zatížením systému, a proto je nutné, aby i tento generátor se tímto způsobem choval. Z tohoto důvodu generátor vytěžuje aplikaci 40 % z celkového počtu uživatelů.

Registrovaní a přihlášení uživatelé

Druhou skupinou jsou registrovaní a přihlášení uživatelé. Tito uživatelé můžeme dále rozdělit tímto způsobem na:

- Uživatele s e-shopy, kteří chtějí získat nové zákazníky.
- Uživatelé doporučující partnerské e-shopy za účelem získání provize z objednávek.
- Speciální uživatelé, kteří jsou oběma předchozími typy.
- Administrátoři.

Generátor simuluje pouze první dva typy uživatelů, protože speciální uživatelé vytváří zatížení stejné, jako první dva typy uživatelů, a proto není potřeba reprezentovat i tento typ uživatelů. Dalším důvodem je také to, že i kdybychom měli reálné statistiky vytížení produkční aplikace a mohli si zobrazit jeho vytížení, tak nám tato informace při dostatečném množství požadavků řekne stejné výsledky jako od vytížení prvními dvěma typy uživatelů. Administrátorská část není výkonově testovaná, protože se zde očekává minimální zatížení.

Prvním typem uživatelů jsou ti, kteří vlastní nebo spravují e-shopy pro které chtějí získat nové objednávky. Generátor tento typ uživatelů simuluje tak, aby byla zahrnuta do vytížení co největší

funkční část a nejlépe cely životní cyklus, který probíhá pro tento typ uživatele. V simulaci je tento průběh obsažen v 10 % z celkového počtu uživatelů, protože se tento průběh neočekává až tak příliš často.

Jednotlivé kroky simulace pro uživatele s e-shopy:

1. Přihlásí uživatele.
2. Vytvoří nový e-shop.
3. Zobrazí jednotlivé objednávky e-shopu.
4. Zobrazí jednotlivé návštěvy e-shopu.
5. Upraví e-shop.
6. Odstraní e-shop.
7. Odhlásí uživatele.

Druhým typem uživatelů jsou pak ti, kteří doporučují jednotlivé e-shopy s cílem získat určitou provizi za uskutečněný nákup. U tohoto typu je zatížení generováno opět tak, aby se toto zatížení co nejvíce blížilo reálnému. V zatížení je tento průběh obsažen v 20 % případů.

Jednotlivé kroky simulace pro uživatele doporučující e-shopy:

1. Přihlásí uživatele.
2. Zobrazí svůj přehled.
3. Zobrazí své e-shopy.
4. Zobrazí si oblíbené e-shopy.
5. Zobrazí všechny e-shopy.
6. Náhodně si vybere e-shop a zobrazí jeho všechny bannery.
7. Vrátí se zpět na výpis všech e-shopů.
8. Přidá náhodný e-shop do oblíbených.
9. Odebere e-shop z oblíbených.
10. Odhlásí uživatele.

Neregistrovaní uživatelé

Třetí skupinou jsou neregistrovaní uživatelé, kteří přistupují na volně dostupné časti. Tito uživatelé jsou ještě dále rozdělené na tyto skupiny:

1. Postupné zobrazení všech veřejných stránek.
2. Přístup na náhodnou stránku s přístupem přes hlavní stránku.
3. Přečtení všech článků z blogu.

Každá tato skupina je v zatížení obsažena v 10 %. Toto zatížení bylo zvoleno proto, že takové zatížení by se v reálném nasazení mohlo objevit, a to z důvodu zatížení tvořeného čtením jednotlivých článků na blogu nebo díky zátěži od nových potenciálních zákazníku, kteří se o tuto službu zajímají. Dalším menším důvodem pro takové rozložení bylo zbývající procentuální rozložení celkového počtu uživatelů pro simulaci, kterých zbývalo posledním 30 %, a které se pro takové rozložení přímo nabízelo. Z jednotlivých názvů skupin lze jednoduše zjistit, jakým způsobem vytěžují veřejnou část aplikace. Z tohoto důvodu není potřeba jednotlivé skupiny více popisovat. Níže lze pro každou skupinu vidět přesné kroky prováděné v simulaci.

Jednotlivé kroky simulace pro postupné zobrazení všech veřejných stránek:

1. Zobrazí hlavní stránku.
2. Zobrazení postupně všechny články na blogu.
3. Zobrazí stránku (Testovací stránka) vytvořenou přes administraci.
4. Zobrazí povolené e-shopy využívající tuto službu.
5. Navštíví kontaktní stránku.

Jednotlivé kroky simulace pro přístup na náhodnou stránku s přístupem přes hlavní stránku:

1. Zobrazí hlavní stránku.
2. Náhodně vybere jednu stránku, kde v konečném výsledku bude počet navštívení pro každou stránku podobné.
    a. Zobrazí stránku (Testovací stránka) vytvořenou přes administraci.
    b. Zobrazí povolené e-shopy využívající tuto službu.
    c. Zobrazení postupně všechny články na blogu.
    d. Navštíví kontaktní stránku.

Jednotlivé kroky simulace pro Přečtení všech článků z blogu:

1. Zobrazí hlavní stránku.
2. Zobrazí stránku blogu.
3. Zobrazí první článek.
4. Vrátí se zpět na stránku blogu.
5. Zobrazí druhý článek.

Shrnutí

V této podkapitole jsme si navrhli podobu vytížení naší aplikace, které bude tvořeno tímto generátorem, a kde jsem se snažil o vytvoření zatížení, které by se mohlo podobat co nejvíce reálnému vytížení dosaženém v produkční verzi. Kromě toho jsem toto zatížení procentuálně rozložil mezi jednotlivé části webu, které jsem určil podle typu určení služby a předpokládané zátěže. Toto procentuální rozložení lze pro rekapitulaci vidět v tabulce 3. V následující podkapitole pak bude popsaná implementace tohoto generátoru.

Veřejná část aplikace	Neveřejná část aplikace	Přesměrování uživatelů
30 %	30 %	40 %

Tabulka 3: Procentuální rozložení generátoru zatížení

8.3.2 Implementace Generátoru

V této podkapitole navážu na minulou podkapitolu, kde jsem navrhl podobu generátoru. Nyní se pokusím popsat jeho implementaci a problémy se kterými jsem se během implementace setkal. Jak už bylo popsáno výše, tak tento generátor bude využívat nástroj pro výkonové testovaní, kterým je Gatling.

V tomto nástroji je potřeba napsat testy, které budou vytvářet jednotlivou zátěž a budou představovat tento generátor zatížení. Jednotlivé testy se píší v programovacím jazyce Scala, který je funkcionálně-skriptovací jazyk, který vychází z Javy a plně podporuje její knihovny. S tímto programovacím jazykem jsem neměl žádnou předchozí zkušenost. Kromě toho nástroj Gatling nabízí

většinu potřebné funkcionality, která je velmi dobře popsána v jeho dokumentaci. V této dokumentaci najdeme také počáteční návod pro rychlý začátek a mnoho příkladů pro ukázku použití. Také spoustu návodů a ukázkových příkladů lze najít na internetu. Kromě toho Gatling nabízí ještě jeden pomocný nástroj, díky kterému můžeme vytvářet jednotlivé testy, a to pomocí nahraní činnosti ve webovém prohlížeči. Tento pomocný nástroj může velmi pomoci a ulehčit práci při seznamovaní se s tímto nástrojem. Výsledkem této nahrávky je pak soubor obsahující test, který reprezentuje nahranou činnost z webového prohlížeče. Pro tuto funkci je v prohlížeči potřeba pouze nastavit proxy server [41], kterým bude aplikace Gatling. Díky tomu všemu lze celkem jednoduše a rychle vytvořit jednotlivé testy. Spoustu informací ohledně nastavené a konfigurace jsem našel v oficiální dokumentaci a také na jednom zahraničním blogu [42; 43].

Pro jednotlivé testy jsem si vybral složku test/gatling nacházející se v hlavním adresáři aplikace, kde složka test obsahuje testové soubory v tom případě, jestliže používáme pro testovaní naší aplikace funkcionalitu nabízenou přímo frameworkem Ruby on Rails. Ovšem kvůli testovaní aplikace pomocí frameworku Rspec, byla tato složka prázdná, a proto byla vhodným místem pro umístění generátoru. Struktura těchto testů byla přizpůsobená struktuře požadované aplikací maven [44] se závislostí na modulu Gatling. Díky tomuto propojení je zajištěno automatické stažení Gatlingu a také jednoduchá možnost pro programové spouštění testů. Při vytváření této struktury a propojení s mavenen mi pomohly dva zahraniční weby [45; 46]. Strukturu této složky lze vidět níže.

- src
  - resources
    - bodies
    - data
    - affil.csv
  - scala.cz.provizne.gatling
    - elements
    - simulation
- target

V této struktuře jsou zdrojové soubory umístěné ve složce src a jednotlivé výstupy se pak nacházejí ve složce target. Složka resources pak obsahuje strukturu, která je vyžadovaná aplikací Gatling. Popis těchto souborů lze najít v dokumentaci. Kromě této struktury se zde nachází soubor affil.csv, který obsahuje vstupní testovací data, která jsou generovaná pomocí této aplikace. Těmito daty jsou hash klíče, které identifikují uživatele a nějaký náhodný e-shop. Tyto vstupní data se pak následně používají při testovaní přesměrovaní uživatele na daný e-shop. Postup generovaní tohoto souboru bude popsáno později. Ve složce elements jsou pak obsaženy jednotlivé modely obsahující jednotlivé testy. Také zde najdeme model, který obsahuje nastavení použité v našich testech a přebírá určité proměnné z nástroje maven. Ve složce simulation se pak nachází soubory simulace, které jsou spouštěny Gatlingem jako celkový test. Tyto soubory obsahují testové případy obsažené v modelech, které jsou pak volány v celkovém testu. Ve složce target pak najdeme všechny výstupy vytvářené pomocí mavenu nebo výstupy z jednotlivých testů. [47]

```
val reqGetNewEshop = exec(http("Get new eshop form")
  .get("/eshops/new")
  .check(css("h1:contains('Nový Eshop')"))
)

val reqPostCreateEshop = exec(http("Create new eshop")
  .post("/eshops")
  .formParam("utf8", "✓")
  .formParam("eshop[name]", "eshop1")
  .formParam("eshop[url]", "http://example.cz")
  .formParam("eshop[import_address]", "http://xml.example.cz")
  .formParam("eshop[provision]", 5)
  .formParam("eshop[description]", "some description")
  .formParam("commit", "Uložit změny")
)

val createEshop = exec(
  reqGetEshops, reqGetNewEshop, reqPostCreateEshop, reqGetIdCreatedEshop
)

val groupCreateEditDeleleEshop = group("New eshop steps") {
  exec(
    User.reqLoginUser, createEshop, reqGetEshopOrders, …, User.reqLogoutUser
  )
}

val scnCreateEditDeleleEshop = Constants.createScenario("Create edit and delete eshop",
                                    User.userFeeder, groupCreateEditDeleleEshop)
```

Obrázek 23: Ukázka funkcí v testech nástroje Gatling

V této ukázce na obrázku 23 lze vidět většina zajímavých a nejčastěji používaných funkcí v jednotlivých testech. Pokud se podíváme na ukázku tohoto kódu, tak hned vidíme na první pohled, že se jedná o funkcionální skriptovací programovací jazyk.

Úplně na začátku můžeme vidět první test, který získá formulář pro vytvoření nového e-shopu. Tento test je ukázkou pro GET požadavek, kde typ tohoto požadavku je určen pomocí funkce get. Dále zde můžeme vidět funkci check, díky které si můžeme ověřit správnost dat během průběhu testu. V tomto případě je to ověření, zda se na stránce nachází korektní nadpis typu h1. Každý test pak musí ještě obsahovat funkce exec a http. Funkce exec umožnuje spouštění funkcí a funkce http pak definuje název testu, který lze vidět ve výstupech. Tento název musí být unikátní.

Jestliže se pak podíváme na další test, tak můžeme vidět, že vytváří nový e-shop. Tento test je ukázkou pro POST požadavek, který je určen pomocí post metody. Jednotlivé hodnoty, které mají být použity pro vytvoření e-shopu jsou definované pomocí funkce formParam.

Funkce createEshop slouží pouze pro vytvoření skupiny testů, kde pořadím je určena posloupnost jednotlivých testů. Díky takovým typům funkcí, jsem schopen si definovat různé testovací scénáře.

Následná funkce groupCreateEditDeleleEshop pak má stejný význam jako funkce createEshop, ale navíc obsahuje ještě funkci group, díky které si lze vytvořit různé skupiny testů, které pak lze vidět ve výsledcích. V těchto výsledcích pak budou jednotlivé testy zanořeny do této skupiny.

8.3.3 Spouštění testů a ukázka jejich výstupů

Jestliže si chceme spustit zátěžové testy tak potřebujeme mít nejdříve nainstalované potřebné závislosti kterými jsou maven a minimálně Java 8 v JDK verzi. Samotný Gatling není potřeba stahovat, protože se o to postará maven. Jestliže tyto závislosti máme nainstalované tak spustíme naší aplikaci, kterou chceme testovat. Následně před samotným spuštěním je ještě potřeba nastavit proměnně v pom souboru. Například je zde potřeba správně nastavit host a port na kterém běží aplikace, kterou chceme testovat. Dále zde můžeme povolit ladění, nebo nastavit počet simulovaných uživatelů.

Jestliže všechny závislosti máme splněny tak můžeme přejít ke spuštění našich testů. Pro toto spuštění musíme být v kořenovém adresáři, kde se nachází testy. V našem případě je to složka test/gatling, kterou lze najít v kořenové složce aplikace. Jestliže se již v této složce nacházíme, tak spuštění těchto testů je díky mavenu velice jednoduché, a to pomocí tohoto příkazu:

```
mvn test
```

Po spuštění tohoto příkazu, maven nejdříve zkontroluje všechny jeho závislosti, eventuálně je stáhne a nainstaluje. Poté začne spouštět testy s nastaveným počtem simulujících uživatelů. Průběh testu je zobrazen v konzoli a po jeho dokončení je vygenerovaný HTML výstup, který lze nalézt ve složce target. Jednotlivé ukázky výstupu generátoru lze vidět níže. Tato ukázka byla vytvořena pro 50 uživatelů a obsahuje téměř celý grafický výstup. Díky tomu, že je to HTML výstup, tak grafy nabízí interakci při nájezdu myší a zobrazí konkrétní hodnoty pro tento bod. V globálních informacích se nachází informace o průběhu testu. Na začátku je grafický výstup, který je na obrázku 24, obsahující přehled celkového počtu požadavků proti době vykonání a počtu neúspěšných požadavků. Pod tímto grafickým výstupem se pak nachází celkový přehled, který obsahuje důležité hodnoty pro všechny požadavky. Mezi tyto hodnoty například patří počet dotazů, požadavky/sec, počet úspěšných nebo selhaných požadavků, doba odpovědi a další. Kompletní výstup lze vidět v ukázce na obrázku 25.

Obrázek 24: Výstup z generátoru-Globální graf

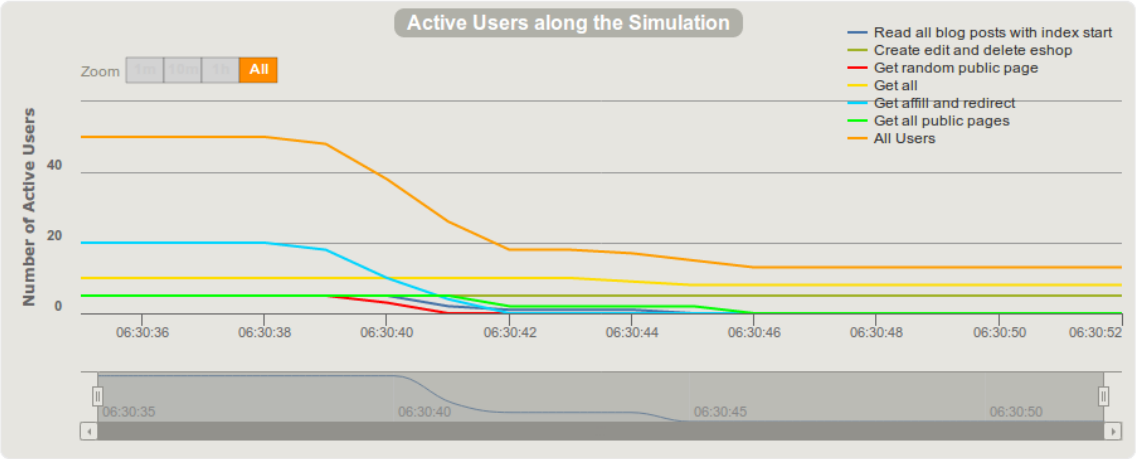
ASSERTIONS

Assertion	Status
Global: max of response time is less than or equal to 800.0	KO
Global: percentage of successful requests is greater than or equal to 99.0	OK

STATISTICS (Click here to show more) — Expand all groups | Collapse all groups

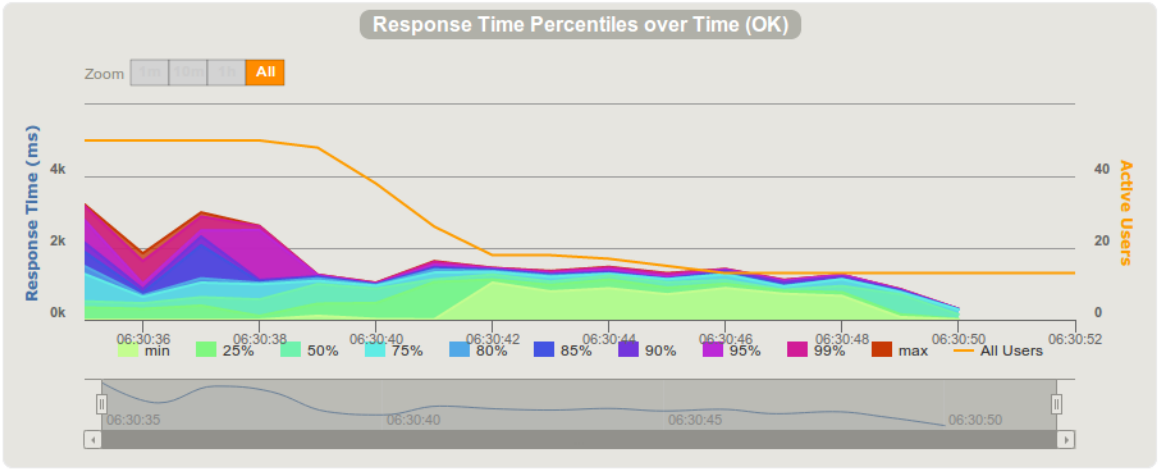
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	318	318	0	0%	17.667	2	859	1115	1943	2906	3216	843	585
Go throu...ic pages	5	5	0	0%	0.278	2731	2885	6753	7007	7058	7071	4459	2006
Random p...ex start	5	5	0	0%	0.278	800	1415	1460	1944	2041	2065	1312	470
Basic user steps	10	10	0	0%	0.556	5923	16727	17193	18675	18773	18798	15011	4465
Read blog posts	5	5	0	0%	0.278	2018	2348	2351	5180	5746	5887	2950	1474
Get index	5	5	0	0%	0.278	322	422	423	1626	1867	1927	700	614
Get blog contents	10	10	0	0%	0.556	371	539	566	1023	1024	1024	595	224
Get blog article 1	5	5	0	0%	0.278	292	434	576	922	991	1008	529	259
Get blog article 2	5	5	0	0%	0.278	391	449	474	820	889	906	530	190
New eshop steps	5	5	0	0%	0.278	16031	17382	17970	18575	18696	18726	17318	977
Get affill page	20	20	0	0%	1.111	207	1210	2101	3155	3204	3216	1391	987
Get affi...direct 1	20	20	0	0%	1.111	2	3	7	9	10	10	4	3

Obrázek 25: Výstup z generátoru-Globální výsledky

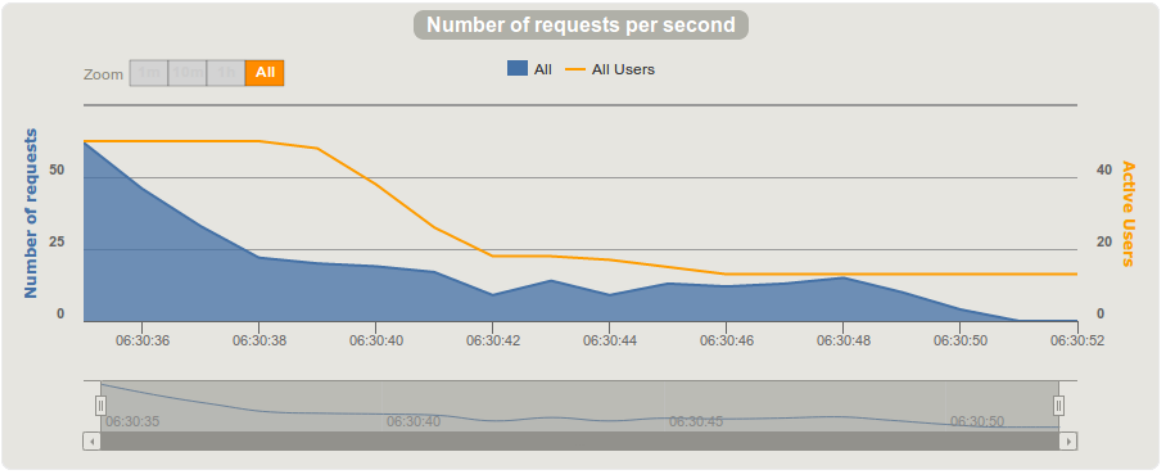


Obrázek 26: Výstup z generátoru-Graf s aktivními uživateli

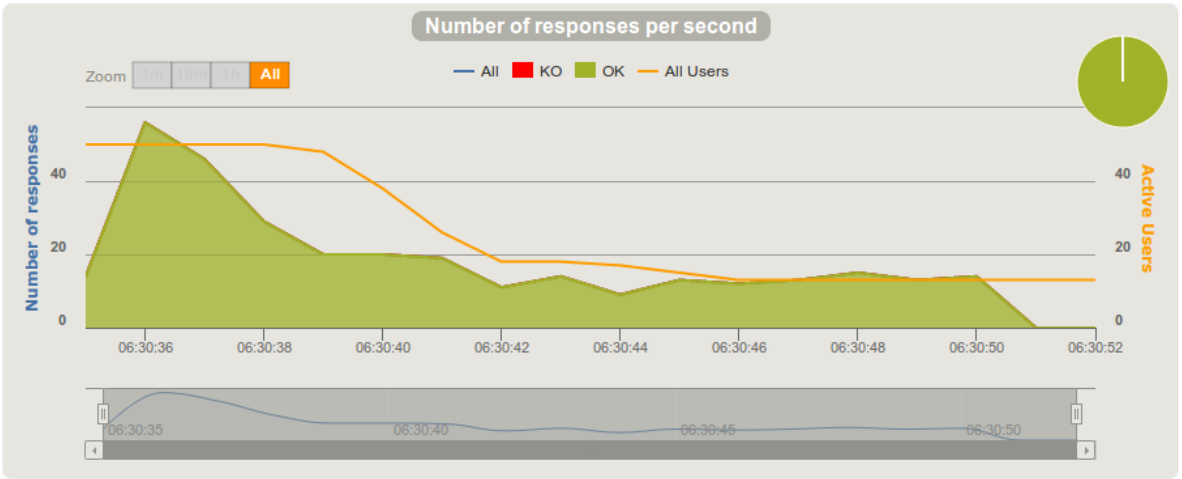
Po globálních informacích obsahuje výstup několik grafů, které jsou zaměřeny na konkrétní typ výstupu. Jeden z grafů na obrázku 26 obsahuje aktivní uživatele během simulace vůči času. Díky tomuto grafu vidíme, kdy jednotliví uživatele dokončili svůj průběh. Následující graf na obrázku 27 zobrazuje rozložení odpovědí úspěšných požadavků vůči času v percentilovém tvaru. Také zobrazuje rozložení počtu uživatelů. V dalším grafu na obrázku 28 je pak počet požadavků za sekundu v průběh času. Poslední graf z obrázku 29 pak zobrazuje počet odpovědí v časovém průběhu.



Obrázek 27: Výstup z generátoru-Graf odpovědí serveru v čase v percentilovém tvaru



Obrázek 28: Výstup z generátoru-Graf s počtem požadavků



Obrázek 29: Výstup z generátoru-Graf odpovědí serveru v čase

9 Optimalizace systému

Z důvodu předpokládané vysoké zátěže na určité části aplikace bylo potřeba se pokusit aplikaci a systém co nejvíce optimalizovat. Toto očekávané vytížení systému jsou statisíce paralelních přístupů. Z této kapitoly a také bodu zadání mé diplomové práce jsem měl největší počáteční strach, protože jsem neměl žádné předchozí zkušenosti s řešením této problematiky. Nakonec se tento strach ukázal jako zbytečný, jelikož pokud se použijí vhodné nástroje, které jsou pro to určeny a které jsem popsal v minulé kapitole, tak nabízí obrovskou pomoc a podporu pro řešení této problematiky. Díky těmto nástrojům lze pak udělat základní optimalizaci i bez velkých zkušeností.

9.1.1 Optimalizace vytváření testovacích dat

V aplikaci používám při vytváření a migraci databáze generátor ukázkových dat. Mezi tyto data patří například uživatelé, kteří vlastní e-shopy, které obsahují nějaké objednávky. Pro toto vytváření jsem původně použil základní přístup, který lze vidět v ukázce na obrázku 30

```
def self.banners
  Eshop.all.each do |e|
    5.times {
      banner = e.banners.new(name: Faker::Lorem.sentence(1), url: e.url,
                             alt: Faker::Lorem.sentence)
      banner.add_promo
      banner.create
    }
  end
end
```

Obrázek 30: Ukázka neoptimalizovaného vytváření testovacích e-shopů

V obrázku 30 jde vidět úzké hrdlo, které je způsobené tím, že pro každý e-shop je vytvořeno pět bannerů, které jsou vytvářeny postupně každý zvlášť. Tímto způsobem je pro každé vytvoření banneru použitý jeden SQL dotaz, kde tímto vznikne velké množství dotazů na databázi. Tento způsob je časově náročný pro větší počet dat. Jako řešení toho úzkého hrdla mě okamžitě napadlo použít hromadné SQL dotazy. Problém ovšem byl takový, že pokud chci používat Active Record [48], tak ten nic takového defaultně nenabízí. Začal jsem tedy hledat na internetu a našel jsem několik možností na službě stackoverflow [49]. Tato webová služba slouží programátorům pro sdílení jejich problémů nebo otázek, kde ostatní uživatele se snaží pomoci.

Jako první řešení jsem našel použití pole, do kterého vložíme všechny záznamy, které chceme vytvořit. Následně je pak vytvoříme pomocí funkce create pro daný model. Ovšem toto řešení nemělo požadovaný výsledek, jelikož i toto řešení vytvořilo pro každý objekt v poli jeden SQL dotaz.

Další nalezenou možností bylo použití transakcí. Toto řešení se mi nelíbilo, protože podle mého názoru, by se transakce měli používat na operace, kde potřebujeme zajistit, aby proběhli všechny části bez chyby, nebo jsme se vrátili do původního stavu před touto operací.

Poslední nalezené východisko řešilo tento problém tak, jak jsem požadoval, a to pomocí použití hromadných dotazů, které byly přidány jako nová funkcionalita do Active Record pomocí gemu activerecord-import [50]. Toto bylo to, co jsem se snažil najít, a nakonec i našel v tomto rozšíření. Po úspěšné instalaci tohoto gemu jsem upravil celý proces pro vytváření ukázkových dat. Na ukázce níže lze vidět tato úprava pro vytváření bannerů.

```
def self.banners
  banners = []
  Eshop.all.each do |e|
    5.times {
      banner = e.banners.new(name: Faker::Lorem.sentence(1), url: e.url, alt:
Faker::Lorem.sentence)
      banner.add_promo
      banners << banner
    }
  end
  Banner.import banners, :validate => false
end
```

Obrázek 31: Ukázka optimalizovaného vytváření testovacích e-shopů

V ukázce na obrázku 31 lze vidět, že nyní jsou jednotlivé záznamy nejdříve před-vytvořeny do pole a následně vytvořeny jedním SQL dotazem pomocí funkce import, kterou zde přidal activerecord-import gem. Také u této funkce jsou vypnuté validace, díky kterým se vytváření ještě mírně urychlí. Vypnutí validací jsem si v tomto případě mohl dovolit, protože vytvářím data, o kterých vím, že jsou v korektní podobě.

Pro zhodnocení této optimalizace jsem udělal jednoduchý test, kde jsem nechal vytvořit ukázková data pro 10 uživatelů a změřil jsem potřebnou dobu. Pro zpřesnění výsledku jsem provedl celkem tři měření a vypočetl jsem průměrnou dobu, která je uvedena ve výsledcích. Testování probíhalo na mém počítači. Výsledky lze vidět v tabulce 4.

	Doba zpracování (s)	
	S validací	Bez validace
Původní vytváření dat	92,85	-
Optimalizované vytváření dat	7,74	6,42

Tabulka 4: Výsledky původního a optimalizovaného vytváření testovacích dat

Z tohoto výsledku jde jasně vidět, že optimalizace mnohonásobně zvýšila rychlost vytváření testovacích dat. Pokud k tomu ještě vypneme validace, tak získáme mírné zlepšení, které v mém případě bylo o 1,3 sekundy lepší. Každý si sice může říct, že toto zlepšení není nijak výrazné, ovšem při větším množství dat, toto vypnutí validací ušetří nějakou dobu, která je již výrazná.

9.1.2 Optimalizace aplikace:

V této podkapitole se pokusím o optimalizaci aplikace pomocí grafického nástroje, pomocí kterého budu postupně procházet jednotlivé části služby a hledat v nich problémy. Pro optimalizaci jsem využil nástroj Rack-miniprofiler, který jsem už popsal v podkapitole, která se věnovala hledání a výběru nástrojů pro ulehčení optimalizace. Během této optimalizace jsem kladl důraz na dobu zobrazení webové stránky, u které jsem vyžadoval, aby tato doba nepřekročila 250ms. Dalším parametrem, na který jsem se zaměřil, byly SQL dotazy volané pro každou stránku. U těchto dotazů mě zajímal jejich počet, u kterého jsem přemýšlel, zda bych některé nemohl nahradit, nebo úplně zrušit. Kromě toho mě také zajímala jejich doba provedení. Pokud tato doba byla vysoká, nebo byla v částí s předpokládanou vysokou zátěží, tak jsem se je snažil optimalizovat. Tato optimalizace pak probíhala tak, že jsem si pro tento dotaz zobrazil plán vykonávaní dotazů a pokud bylo potřeba, tak jsme zkoušel přidat rozumně

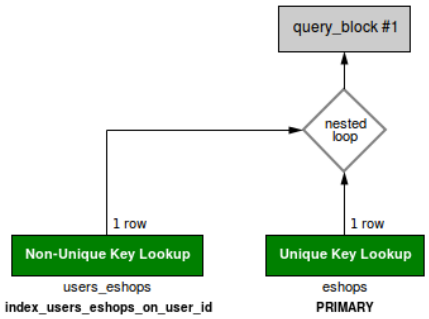
index, který by tento dotaz zlepšil. Poslední parametr, na který jsem se zaměřil, byla doba potřebná pro vykreslení šablony.

Při této optimalizaci jsem nenarazil na moc problémů, protože většina stránek splnila moji podmínku se zobrazením do 250ms a obsahovala rozumné množství dotazů na databázi. Tento výsledek mohl být způsoben díky tomu, že optimalizace probíhala na mém vývojovém stroji, kde server nebyl vůbec zatížen a zpracovával pouze moje požadavky a také tím, že vytváření jsem kladl důraz na vytváření indexů na parametrech, které budou často použity pro vyhledávaní.

Během této optimalizace jsem zjistil, že nebyly použity cizí klíče, které framework Ruby on Rails do verze 4.2 automaticky nepodporoval. Dále jsem upravil několik dotazů, kde jsme přidal potřebné indexy, které zlepšily výkon dotazu. Nakonec jsem provedl přepsaní jednoho složitějšího dotazu na proceduru, díky které bylo dosaženo mírného zlepšení. Tento dotaz umožnuje vyhledaní všech relevantních informací určených pro přehled uživatele, jak si stál s doporučováním nových uživatelů.

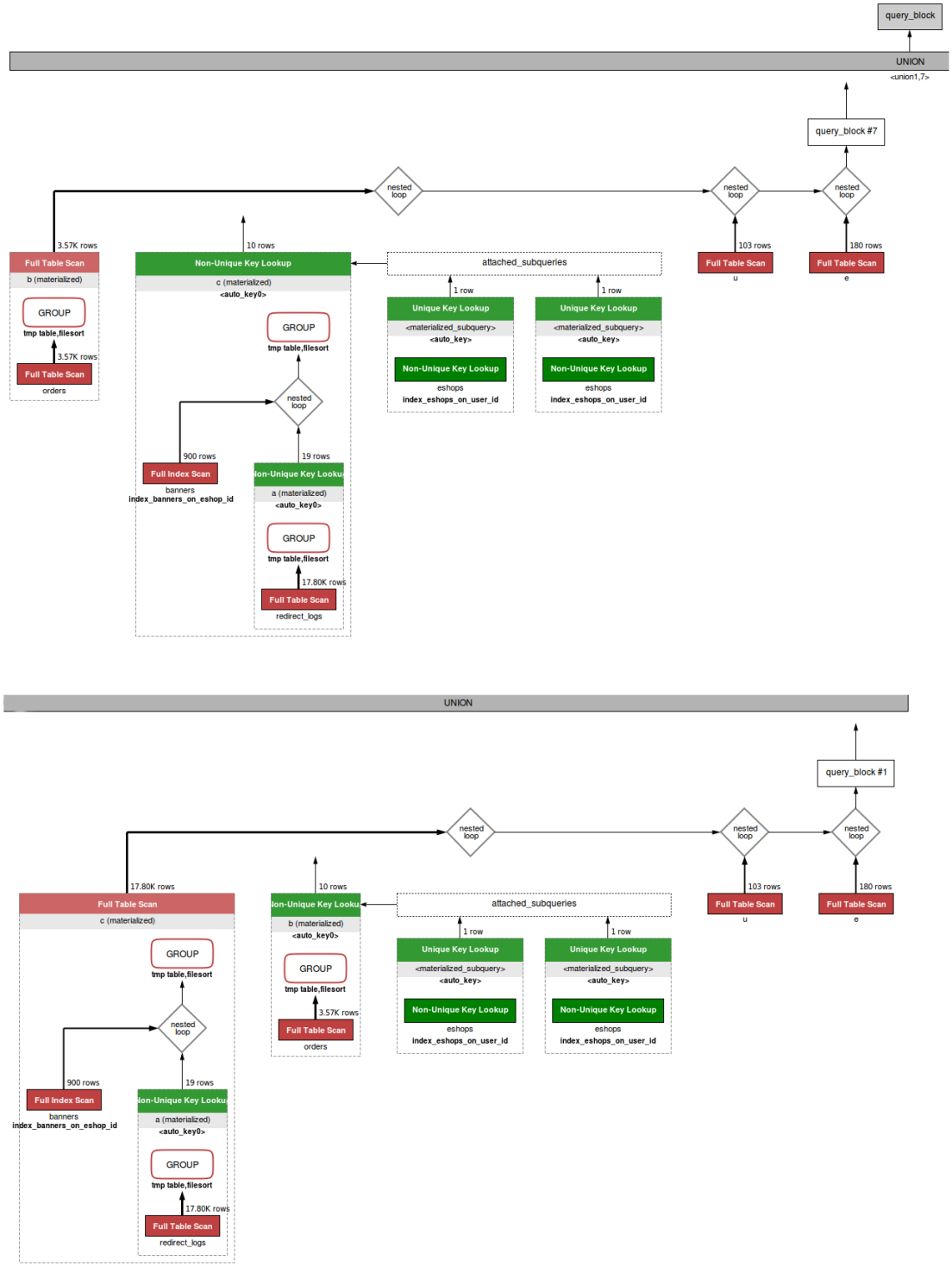
Pro ukázku vykonávaní plánu dotazů, které byly použity během optimalizace vybraných dotazů, jsem vybral dvě ukázky. První ukázka je jednoduchý plán představující výběr oblíbených e-shopů pro konkrétního uživatele. Druhým plán představuje nejsložitější dotaz v databázi, kterým je dotaz pro získání dat pro přehled uživatele.

Z první ukázky na obrázku 32 jde vidět, že tento plán využívá pro vyhledaní jednotlivých záznamů indexy místo prohledaní všech záznamů zvlášť. Díky tomuto přístupu je tento dotaz optimální.



Obrázek 32: Ukázka plánu vykonání dotazu pro získaní oblíbených e-shopů daného uživatele

Na druhém plánu na obrázku 33 můžeme vidět, že jsou využívaný jak indexy, tak fullscany. Tyto fullscany všemi záznamy jsou způsobeny zjišťováním počtu objednávek, počtu přesměrování nebo součtem všech cen objednávek pro určení provize k vyplacení, u kterých je nutné projít všechny záznamy. Vyhodnocení dotazu by tak mělo být optimální.



Obrázek 33: Ukázka plánu vykonání dotazu pro získaní všech informací potřebných pro uživatelský přehled stavu doporučení

9.1.3 Optimalizace aplikačního serveru

Jako aplikační server jsem vybral pro tuto aplikaci pumu [51], jelikož podává celkem dobré výsledky a její licence je opensource. Také je to jeden z nejpoužívanějších serverů. Kromě pumy ještě se používá Unicorn [52] nebo Phusion Passenger [53]. Unicorn jsem chtěl taky vyzkoušet, ale bohužel se mi to nepovedlo, protože mi instalace skončila s chybou, která se mi nepodařila vyřešit.

Pro získaní maximálního výkonu je potřeba vhodně nastavit Pumu, kde lze nakonfigurovat minimální a maximální počet vláken a také počet jednotlivých procesů neboli „workers“. Počet „workerů“ by měl být podle doporučení a taky internetových diskuzí stejný jako počet jader procesoru. Ovšem nikde se už nepíše, jestli počtem jader je myšlený počet fyzických jader nebo virtuálních. Po několika měření jsem zjistil že tento počet by měli být fyzická jádra, protože pokud tento počet více překročíme, tak získáme od aplikačního serveru horší výsledky. Dále se mi měřením podařilo zjistit, že je potřeba mít nastavený dostatečný počet vláken, jinak Puma zvládne menší počet požadavků. Dále se nesmí zapomenou při nastavovaní vláken, procesů nebo při použití konfigurační souboru použít parametr nebo funkci v konfiguračním souboru pro znovunačtení konfigurace. Tuto funkci nebo parametr lze najít v dokumentaci. Pokud bychom toto neprovedli, tak se aplikační server může začít chovat nestabilně [54]. V mém případě to bylo stálým načítáním webové stránky. Jednotlivé výsledky měření lze vidět v tabulkách 5-8. Měření probíhalo tak, že test byl spuštěn bez opakovaní a pouze s jedním typem simulace, která přistupuje na hlavní stránku a následně náhodně navštíví jinou veřejnou stránku aplikace.

	w2-t4-16		**w20-t4-16**	
Číslo měření	1	2	3	4
Uživatelé	1000	2000	3000	5000
Počet požadavků	2750	4125	8250	13750
Selhané požadavky	0	323	0	0
Požadavky/sec	47,4	55	211,5	221,77

Tabulka 5: Naměřené výsledky pro nastavení pumy, část 1

	w20-t4-16	**w30-t4-16**	**w40-t4-16**	**w40-t1-1**
Číslo měření	5	6	7	8
Uživatelé	8000	6000	8000	5000
Počet požadavků	22000	16500	22000	13750
Selhané požadavky	1277	666	8537	1543
Požadavky/sec	285,7	235,7	285,7	163

Tabulka 6: Naměřené výsledky pro nastavení pumy, část 2

	w20-t4-25	**w20-t4-45**	**w20-t20-20**	
Číslo měření	9	10	11	12
Uživatelé	8000	8000	8000	8000
Počet požadavků	22000	22000	22000	22000
Selhané požadavky	2963	3438	3230	2854
Požadavky/sec	229,16	231,57	234	231,58

Tabulka 7: Naměřené výsledky pro nastavení pumy, část 3

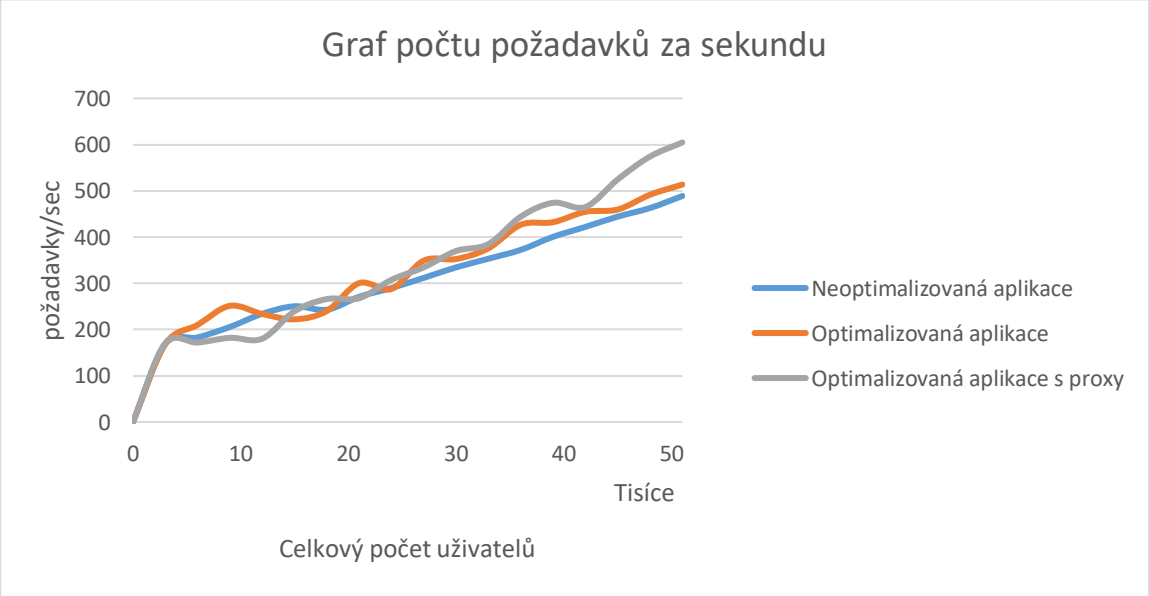
	w20-t35-35	w20-t0-35	w20-t4-16
Číslo měření	13	14	15
Uživatelé	8000	8000	8000
Počet požadavků	22000	22000	22000
Selhané požadavky	3981	2124	3452
Požadavky/sec	207,55	250	224

Tabulka 8: Naměřené výsledky pro nastavení pumy, část 4

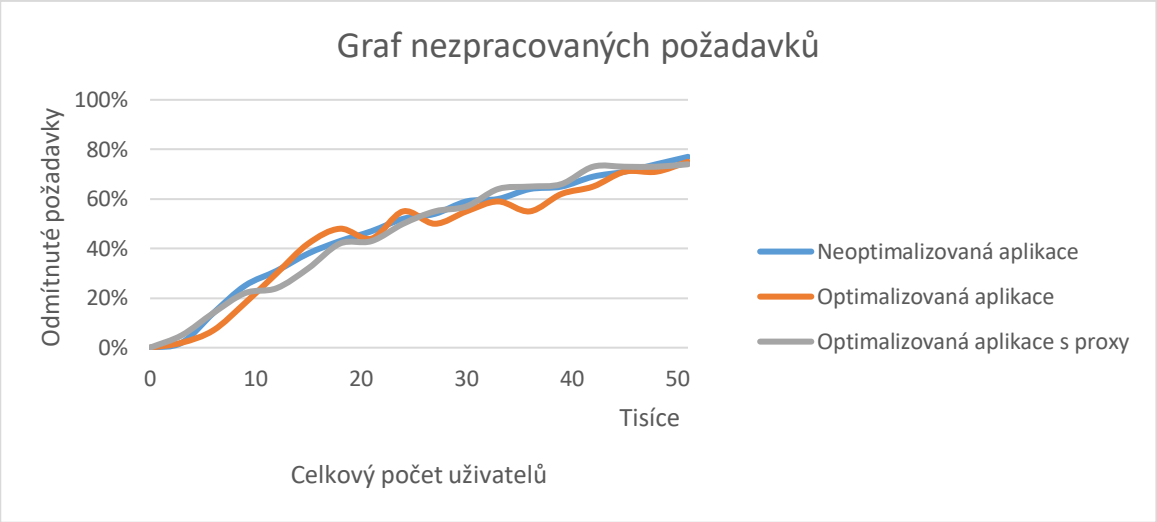
V měření jsou jednotlivé sloupce označeny tímto způsobem **wX-tY-Z,** kde w představuje počet procesů/workerů s hodnotou X, t pak představuje počet vláken, kde Y je minimální počet a Z je maximální počet vláken. Z výsledků jednotlivých měření jde vidět, že nejlépe je aplikační server nastaven s 20 procesy, které se velmi blíží k počtu fyzických jader procesorů. Pro použití nastavení serveru vypadá nejlépe nastavení z těchto měření 5, 9-12 a 14, 15, kde sice nejlépe vypadá měření číslo 5, ovšem pokud se podíváme na měření číslo 15, ve kterém je uděláno ověření se stejným nastavení jako u měření číslo 5, tak jde vidět, že je hodně rozdílné, tudíž se tyto hodnoty liší podle aktuálního vytížení serveru. Podle mého názoru v tomto případě zvolení nějaké konkrétní hodnoty z vybraných hodnot až tak zásadně neovlivní výkon aplikace.

10 Zatížení aplikace a možnosti další optimalizace

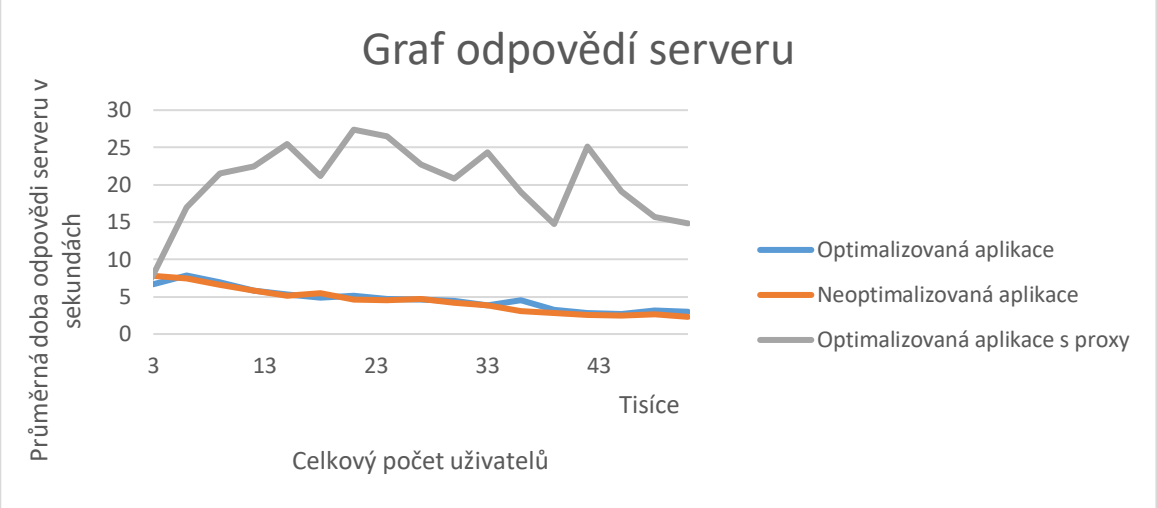
Aplikace byla zatížena pomocí vytvořeného generátoru, kdy se postupně měnil skokově počet uživatelů po 3000. Testování jsem ukončil při 17. měření z důvodu nestíhání obsluhy jednotlivých požadavků, kdy zamítnuté požadavky tvořily přes 75 %. Bohužel se mi nepodařilo dostat na požadované statisícové vytížení dle zadání, protože testovací server by takové zatížení nezvládl. Podařilo se mi dostat na 51000 uživatelů, kteří následně generovali zatížení. Ovšem už při tomto zatížení 75 % požadavků bylo zamítnuto nebo selhalo. Tato data jsou na obrázku 35. Jak lze z naměřených výsledků vidět, nacházející se na obrázku 34, tak optimalizovaná aplikace má mírně lepší výsledky pro počet požadavků za sekundu, kde nelze určit jednoznačně, zda se jedná o zlepšení způsobené optimalizací nebo o různé vytížení testovacího serveru. Pokud se podíváme na optimalizovanou aplikaci s použitím reverzní proxy, tak vidíme, že má velmi podobné výsledky většinu zatížení, jako optimalizovaná aplikace. Její výsledky se začínají měnit od 40 000 uživatelů, kdy aplikace s použitím reverzní proxy má lepší výsledky o 20 % než pouze optimalizovaná aplikace. Můžeme říci, že pokud použijeme reverzní proxy, tak získáme lepší výsledky pro větší zatížení. Pro ostatní sledované vlastnosti nelze s jistou říci, zda tato optimalizace pomohla také ve snížení doby odpovědí jednotlivých požadavků nebo ve snížení počtu odmítnutých požadavku při stejném zatížení, protože tento výsledek nelze z měřených dat přímo určit. Tento nejednoznačný výsledek pro ostatní parametry mohl být způsoben měření dat v různou dobu a také rozdílnou zátěži testovacího serveru. Pokud se podíváme na graf s dobou odpovědí obsažený na obrázku 36, tak lze vidět klesající čas obsluhy jednotlivých požadavku se zvyšujícím počtem uživatelů, který je pravděpodobně způsoben neobsloužením ostatních nezamítnutých požadavků. U aplikace s použitím reverzní proxy, lze vidět očekávané zvyšování doby odpovědi se zvyšujícím se zatížením na začátku grafu, ovšem pak už se to také mění a spíše se doba odpovědi snižuje se zvyšující zátěží.



Obrázek 34: Zatížení aplikace-graf počtu požadavků za sekundu



Obrázek 35: Zatížení aplikace-graf nezpracovaných požadavků



Obrázek 36: Zatížení aplikace-graf odpovědí serveru

10.1 Další možnosti optimalizace

Jako další možností optimalizace by mohlo být použití více aplikačních serverů a následné rozložení zátěže s použitím cloudového DBS nebo distribuovaného na více instancích. Další řešení by mohlo být použití nějakého externího serveru pro cachování dat, který by mohl velmi zlepšit jednotlivé výsledky. Tímto serverem by mohl být například Redis [55]. Jako poslední možnost by pak mohlo být více se zaměřit na optimalizaci databáze, a to konkrétně na správné použití jednotlivých indexů.

11 Porovnání systému

Na trhu existuje mnoho provizních informačních systému, ovšem většinou se jedná pouze o systém, který je přímo součástí konkrétního webového obchodu nebo dané poskytované služby. Díky tomu, systém nabízí provizi pouze z omezeného sortimentu zboží, které je nabízeno pouze daným e-shopem nebo službou. Příkladem je například BIBLOO [56] zaměřující se na módu. Dalším druhem provizního systému jsou ty, které sice nabízejí produkty z více zdrojů, ale jsou omezeny pouze na určitou kategorii zboží. Touto kategorií muže být například poskytovaní zájezdů nebo pojištění. U nás takový velmi rozšířený a známý provizní systém je například Invia [57], která poskytuje velké množství zájezdů od mnoho cestovek a nabízí různé druhy provize za uskutečněnou objednávku. Posledním typem jsou pak ty, které již mají sít různých e-shopů, které propagují zboží z různých kategorií, ovšem těchto systémů není příliš mnoho kvalitních. Mnoho těchto systémů je omezeno pouze na poskytováním odkazu a nenabízí propagační bannery. Další nabízí provizi z určitých produktů, místo z celé objednávky, nebo mají nepřehledné rozhraní. Příkladem systému z této kategorie je to například AffilBox [58], u kterého mi přijde, že mají celkem nepřehledné rozhraní, a navíc je potřeba platit měsíční paušál, díky kterému se tento systém nemusí vyplatit menším e-shopům.

Tento webový provizní systém se od ostatních zmíněných liší tím, že nabízí přehledné rozhraní s jednoduchým ovládáním, provize z celé objednávky z kategorií určených zapojenými e-shopy a bez potřeby platit měsíční paušál, díky kterému se tento systém vyplatí i menším e-shopům. Nejtěžší problém pro začátek použití této aplikace bude získaní dostatečného množství kvalitních partnerů, jak s e-shopy, tak pak ty, kteří následně budou jednotlivé e-shopy doporučovat. Ovšem tento úkol je pro provozovatele této služby a jeho marketingu.

12 Závěr a dosažené výsledky

V této diplomové práci jsem vytvořil webový provizní informační systém dle zadání, které bylo vytvořeno firmou Railsformers s. r. o.. Postupně jsem postupoval dle jednotlivých bodů zadání. Nejdříve jsem provedl specifikaci, kde jsem shrnul a popsal co by tento systém měl umět. Následně jsem analyzoval tyto požadavky a poté návrh řešení systému. V těchto částí jsem například provedl technickou specifikaci, vybral jsem jednotlivé rozměry podporovaných reklamních bannerů, popsal základní princip provizního systému, vytvořil třídní diagram a navrhl rozhraní aplikace. Také jsem zde popsal UTM parametry, k čemu slouží a proč jsou v této aplikaci potřeba.

Následně jsem pak pokračoval dalším bodem zadání, kde jsme provedl rešerši jednotlivých SQL cloudových systémů, kde jsem se zaměřil pouze na ty největší, a nakonec jsem jednu z nich vybral pro produkční nasazení. V této práci jsem si tuto databázi v cloudu bohužel nemohl vyzkoušet, protože jsem k ní neměl žádný předplacený tarif. Ovšem tohle mě nijak neodradilo od toho, abych zjistil všechny potřebné informace potřebné k propojení. Tady jsem zjistil, že cloudová databáze se chová stejně jako lokální, jenom je poskytována jako služba, u které lze jednoduše měnit parametry. Takže pokud bych měl nějaký placený tarif, tak u aplikace stačí pouze změnit parametry pro připojení k této cloudové databázi.

Poté jsem pokračoval vytvořením funkční analýzy, kde jsem popsal všechny důležité funkce, které budou použité v systému. Následně jsem implementoval samotný webový informační systém, kde jsem se nesetkal s žádným problémem, který bych nakonec nebyl schopen vyřešit sám nebo s menší radou od programátorů z firmy. Tento systém obsahuje všechny potřebné funkce pro provizní systém. Těmito funkcemi jsou například umožnění registrace uživatelů, vytváření e-shopů s jednotlivými bannery, podpora objednávek s následnou možností schválení provize, import objednávek z XML souboru dostupném na e-shopu, přesměrování uživatelů na e-shop z provizní adresy a zaznamenání uživatele, který je k této adrese přiřazen, a další. V tomto systému jsem se také snažil vytvořit přehledné rozhraní a plně ho optimalizovat pro mobilní zařízení.

Jakmile jsem měl tento webový informační systém implementován, tak jsem se mohl plně věnovat dalšímu bodu, kterým byla optimalizace. S optimalizací aplikace jsem neměl téměř žádné znalosti, takže jsem se snažil najít o tom co nejvíce informací. Pak jsem hledal nějaké nástroje, které by mi následně mohli s optimalizací pomoci, kde jsem jich několik našel a pro některé se rozhodl. Následně jsem provedl optimalizaci, kde jsem se věnoval jednotlivým dotazům, vytváření testovacích dat, samotné aplikaci, vhodnému nastavení a vhodné struktuře nasazení. Poté jsem vytvořil generátor výkonového zatížení aplikace a následně zatížil aplikaci. Následně jsem porovnal optimalizované a neoptimalizované výsledky, kde šlo vidět, že optimalizovaná aplikace měla o něco lepší výsledky. Sice to není nějak vysoké zlepšení, ale na mé nulové zkušeností s touto problematikou si myslím, že to nejsou špatné výsledky. Během zatížení se mi nepovedlo aplikaci vytížit se statisíci požadavky z důvodu nestíhaní testovacího serveru, který při 51000 uživatelích nezpracoval 75 % požadavků. Proto jsem se rozhodl měření při této zátěži ukončit. Nakonec jsem srovnal tento webový provizní informační systém s ostatními systémy na trhu a vypracoval dokumentaci potřebnou pro nasazení systému, administrátory a nové uživatele.

Také díky této diplomové práci jsem se naučil nové technologie. Těmito technologiemi je Docker, Ruby on Rails, nástroje pro testovaní a Gatling určený pro výkonové zatížení. Naučil jsem se

zde Docker, kde jsem následně tuto aplikaci propojil s ním. Díky použití Dockeru je pak následné nasazení v jakémkoliv prostředí, které podporuje Docker kontejnery, velice snadné. V Ruby on Rails jsem si zlepšil programování a naučil se, jak aplikaci následně nasadit. Dále jsem se naučil aplikaci výkonově zatížit pomocí nástroje Gatling, kde jsem si napsal výkonové testy, které zatěžují celkovou aplikaci. Osvěžil jsem si také celý proces, který probíhá při vytváření aplikace, a kterým je specifikace, analýza a návrh systému, pak následná implementace, testování a výsledné produkční nasazení. Zjistil jsem také, jaké jsou potřeba postupy během optimalizace aplikace, a které nástroje jsou k tomu vhodné. Díky tomu jsem se naučil základy pro optimalizaci a výkonové zatížení a také se seznámil databázemi v cloudu, kde jsem předtím neměl s těmito problematikami žádné zkušenosti.

13 Literatura

1. **Ruby on Rails.** Ruby on Rails. *Ruby on Rails.* [Online] [Citace: 8. 11. 2016.] http://rubyonrails.org/.

2. **Google.** *Google Adsense.* [Online] [Citace: 21. 9. 2016.] https://www.google.com/adsense/.

3. **Sdružení pro internetový rozvoj (SPIR).** Nové standardy online reklamy 2009. *Sdružení pro internetový rozvoj (SPIR).* [Online] 13. 8. 2010. [Citace: 21. 10. 2016.] http://www.spir.cz/nove-standardy-online-reklamy-2009.

4. **Google.** Google Analytics Solutions. [Online] [Citace: 21. 9. 2016.] https://www.google.com/analytics/analytics.

5. **Rostecký, Jiří.** Proč a jak používat UTM parametry? *Mladý Podnikatel.* [Online] 12.. 2 2015. [Citace: 12. 5. 2016.] https://mladypodnikatel.cz/jak-pouzivat-utm-parametry-t16014.

6. **Google.** Urchin Tracking Module (UTM). *Google.* [Online] [Citace: 5. 12. 2016.] https://support.google.com/urchin/answer/28307?hl=en.

7. **Ruby.** *Ruby.* [Online] [Citace: 15. 11. 2016.] https://www.ruby-lang.org/en/.

8. **Eich, Brendan.** JavaScript. *ECMA international.* [Online] 1995. [Citace: 27. 6. 2017.] http://www.ecma-international.org/publications/standards/Ecma-262.htm.

9. **World Wide Web Consortium (W3C).** CSS3. *W3C.* [Online] [Citace: 27. 6. 2017.] https://www.w3.org/Style/CSS/.

10. **Oracle Corporation.** *MySQL.* [Online] [Citace: 10. 5. 2017.] https://www.mysql.com/.

11. **Twitter Inc.** *Bootstrap.* [Online] 2010. [Citace: 4. 9. 2016.] http://getbootstrap.com/.

12. **Zurb, Inc.** *Zurb Foundation.* [Online] [Citace: 4. 9. 2016.] http://foundation.zurb.com/.

13. **The PostgreSQL Global Development Group.** *PostgreSQL.* [Online] 1996. [Citace: 10. 5. 2017.] https://www.postgresql.org/.

14. **Google.** Cloud SQL. *Google Cloud Platform.* [Online] [Citace: 10. 5. 2017.] https://cloud.google.com/sql/.

15. —. Cloud SQL for MySQL Features. *Google Cloud Platform.* [Online] 27. 4. 2017. [Citace: 10. 5. 2017.] https://cloud.google.com/sql/docs/mysql/features.

16. **Amazon Inc.** *Amazon Aurora.* [Online] [Citace: 10. 5. 2017.] https://aws.amazon.com/rds/aurora/.

17. **Oracle Corporation.** *Oracle Database.* [Online] [Citace: 10. 5. 2017.] https://www.oracle.com/database/index.html.

18. **Microsoft Corporation.** *SQL Server.* [Online] [Citace: 10. 5. 2017.] https://www.microsoft.com/cs-cz/sql-server/sql-server-2016.

19. **MariaDB Foundation.** *MariaDB.* [Online] [Citace: 10. 5. 2017.] https://mariadb.org/.

20. **Amazon Inc.** Amazon RDS for MySQL. *Amazon.* [Online] [Citace: 10. 5. 2017.] https://aws.amazon.com/rds/mysql/.

21. —. DB Instance Class. *Amazon.* [Online] [Citace: 10. 5. 2017.] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html.

22. —. Amazon RDS for MySQL Product Details. *Amazon.* [Online] [Citace: 10. 5. 2017.] https://aws.amazon.com/rds/mysql/details/.

23. **Docker Inc.** *Docker.* [Online] [Citace: 10. 1. 2017.] https://www.docker.com/.

24. **Vrána, Jakub.** Adminer. *Adminer.* [Online] [Citace: 9. 4. 2017.] https://www.adminer.org/cs/.

25. **The Apache Software Foundation.** Apache HTTP Server Project. *Apache.* [Online] [Citace: 10. 4. 2017.] https://httpd.apache.org/.

26. **ruby-prof.** ruby-prof. *Github.* [Online] [Citace: 10. 1. 2017.] https://github.com/ruby-prof/ruby-prof.

27. **Dymo, Alexander.** Make Your Ruby/Rails App Fast: Performance And Memory Profiling Using ruby-prof and KCachegrind. *acunote.* [Online] 6. 2. 2008. [Citace: 26. 4. 2017.] http://www.acunote.com/blog/2008/02/make-your-ruby-rails-applications-fast-performance-and-memory-profiling.html.

28. **MongoDB, Inc.** MongoDB. [Online] [Citace: 27. 6. 2017.] https://www.mongodb.com/.

29. **Saffron, Sam.** Flamegraph. *Github.* [Online] [Citace: 10. 1. 2017.] https://github.com/SamSaffron/flamegraph.

30. **Jones, Richard, Hosking, Antony a Moss, Eliot.** *The garbage collection handbook: the art of automatic memory management.* Boca Raton, FL : CRC Press, c2012. ISBN 1420082795.

31. **Gupta, Aman.** stackprof. *Github.* [Online] [Citace: 5. 1. 2017.] https://github.com/tmm1/stackprof.

32. **Saffron, Sam a přispěvatelé.** memory_profiler. *Github.* [Online] [Citace: 10. 1. 2017.] https://github.com/SamSaffron/memory_profiler.

33. **Saffron, Sam.** rack-mini-profiler. *GitHub.* [Online] 2013. [Citace: 10. 1. 2017.] https://github.com/MiniProfiler/rack-mini-profiler.

34. **Baker, Steven.** RSPEC. *RSPEC.* [Online] [Citace: 8. 10. 2016.] http://rspec.info/.

35. **Ruby on Rails.** Performance Testing Rails Applications. *Github.* [Online] [Citace: 20. 1. 2017.] https://github.com/rails/rails-perftest.

36. **Jonatan Heyman, Carl Byström, Joakim Hamrén, Hugo Heyman.** *Locust.* [Online] [Citace: 5. 1. 2017.] http://locust.io/.

37. **Python Software Foundation.** Python. *Python.* [Online] Python Software Foundation. [Citace: 10. 5. 2017.] https://www.python.org/.

38. **Apache Software Foundation.** Apache JMeter™. *Apache.* [Online] Apache Software Foundation. [Citace: 18. 1. 2017.] http://jmeter.apache.org/.

39. **Gatling Corp.** Gatling. *Gatling.* [Online] [Citace: 10. 1. 2017.] http://gatling.io/.

40. **Scala.** *Scala.* [Online] [Citace: 20. 5. 2017.] https://www.scala-lang.org/.

41. **Ari Luotonen, CERN; Kevin Altis, Intel.** World-Wide Web Proxies. *W3C.* [Online] 24. 5. 1994. [Citace: 28. 5. 2017.] https://www.w3.org/History/1994/WWW/Proxies/.

42. **Gatling Corp.** Documentation. *Gatling.* [Online] [Citace: 10. 1. 2017.] http://gatling.io/documentation/.

43. **Rijn, Roy van.** Getting to grips with Gatling: Custom Feeder. *Royvanrijn.* [Online] 15. 12. 2015. [Citace: 19. 2. 2017.] http://royvanrijn.com/blog/2015/12/getting-to-grips-with-gatling/.

44. **The Apache Software Foundation.** Welcome to Apache Maven. *Apache.* [Online] [Citace: 10. 3. 2017.] https://maven.apache.org/.

45. **Latinov, Lyudmil.** Performance testing with Gatling. *Automation Rhapsody.* [Online] 9. 1 2017. [Citace: 28. 1. 2017.] http://automationrhapsody.com/performance-testing-with-gatling/.

46. **Baeldung.** Intro to Gatling. *Baeldung.* [Online] 25. 6. 2016. [Citace: 5. 3. 2017.] http://www.baeldung.com/introduction-to-gatling.

47. **Engineering, Treselle.** Parameterization in Gatling Stress Test Tool. *Treselle Systems.* [Online] [Citace: 10. 1. 2017.] http://www.treselle.com/blog/parameterization-in-gatling-stress-test-tool/.

48. **Ruby on Rails.** Active Record Basics. *RailsGuides.* [Online] [Citace: 27. 6. 2017.] http://guides.rubyonrails.org/v4.2/active_record_basics.html.

49. **stackoverflow.** stackoverflow. *stack overflow.* [Online] [Citace: 26. 12. 2016.] http://stackoverflow.com/.

50. **Dennis, Zach.** activerecord-import gem. *Github.* [Online] [Citace: 22. 4. 2017.] https://github.com/zdennis/activerecord-import.

51. **Phoenix, Evan a Contributors.** *Puma.* [Online] [Citace: 10. 4. 2017.] http://puma.io/.

52. **Unicorn.** unicorn: Rack HTTP server for fast clients and Unix. *Unicorn.* [Online] [Citace: 20. 1. 2017.] https://bogomips.org/unicorn/.

53. **Phusion Holding B.V.** Passenger. *Passenger.* [Online] Phusion Holding B.V. [Citace: 20. 1. 2017.] https://github.com/phusion/passenger.

54. **Phoenix, Evan a Contributors.** Puma. *Github.* [Online] [Citace: 10. 4. 2017.] https://github.com/puma/puma.

55. **Redis.** *Redis.* [Online] [Citace: 25. 1. 2017.] https://redis.io/.

56. **Digital People, a.s.** BIBLOO. [Online] [Citace: 25. 6. 2017.] https://www.bibloo.cz/affiliate.

57. **Invia.cz, a.s.** Affilate program Invia. *Invia.* [Online] [Citace: 25. 6. 2017.] https://www.invia.cz/affiliate-program/.

58. **OLYMPIC s.r.o.** AffilBox. *AffilBox.* [Online] [Citace: 25. 6. 2017.] https://www.affilbox.cz/.

59. **Scout.** StackProf: The Holy Grail of Rails Profiling. *Scout.* [Online] [Citace: 5. 1. 2017.] http://blog.scoutapp.com/articles/2015/09/16/profiling-rails-with-stackprof.

60. **Schmidt, Maik.** XmlSimple - Easy API to maintain XML (especially configuration files). *Github.* [Online] 2013. [Citace: 21. 10. 2016.] https://github.com/maik/xml-simple.

61. **Šípoš, Juraj.** Cron – správca úloh. *LinuxEXPRES.* [Online] 15. 10. 2007. [Citace: 5. 3. 2017.] https://www.linuxexpres.cz/praxe/cron-spravca-uloh.

62. **Makhmali, Javan.** Whenever. *Github.* [Online] [Citace: 6. 10. 2016.] https://github.com/javan/whenever.

63. **Wikipedia.** *The garbage collection handbook: the art of automatic memory management.* Boca Raton, FL : CRC Press, c2012. ISBN 1420082795.

Seznam příloh

I. dokumentace_administratorska.docx, Administrátorská dokumentace. Elektronická příloha.
II. dokumentace_systemova.docx, Systémová dokumentace, Obsahuje dokumentaci ohledně spouštění aplikace, funkčních a výkonových testů. Elektronická příloha.
III. dokumentace_uzivatelska.docx. Uživatelská dokumentace. Obsahuje návod pro seznámení nového uživatele s aplikací. Elektronická příloha.
IV. zdrojove_kody, Zdrojové kódy. Složka obsahující zdrojové kódy pro optimalizovanou a neoptimalizovanou verzi aplikace. Elektronická příloha.
V. gatling_data.xlsx, Gatling data. Obsahuje přepsaná naměřená data z gatlingu. Elektronická příloha.